ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

EASYDATA: COMPONENTE PARA LA GENERACIÓN AUTOMÁTICA DE LINKED DATA EN PROYECTOS RUBY ON RAILS

DEPARTAMENTO: LENGUAJES Y SISTEMAS INFORMÁTICOS

DIRECTOR DEL PROYECTO: JUAN MANUEL DODERO BEARDO

AUTOR DEL PROYECTO: JUAN VÁZQUEZ MURGA

Cádiz, Febrero 2012

Fdo: Juan Vázquez Murga

Agradecimientos

Este proyecto va dedicado a mi padre y a mi abuelo quienes siempre confiaron y creyeron en mí y me enseñaron el camino que me ha llevado hasta donde estoy hoy.

A mi familia que siempre me han apoyado y han estado en cada momento que los he necesitado.

A Isaura, por todos los largos días de programación que hemos compartido aprendiendo juntos y por ofrecerme tu compañía, consejo y apoyo en todo momento.

Y Juan Manuel, por todo el apoyo y los consejos que me ha dado desde que comencé con este proyecto del que espero que no sea el último en el participemos.

Gracias a todos.

Índice general

Índice de figuras II

Índice de cuadros III

1 Introducción 1
- 1.1 Introducción . . . 1
- 1.2 Objetivos . . . 2
  - 1.2.1 Ingeniería inversa del modelo de datos de la aplicación . . . 5
  - 1.2.2 Administración de la información RDF asociada al modelo de datos para su codificación. . . . 5
  - 1.2.3 Control de acceso a la información mediante RBAC . . . 6
  - 1.2.4 Interfaz de acceso a los *Linked Data* publicados mediante URI's desreferanciable . . . 6
- 1.3 Alcance . . . 6
  - 1.3.1 Competencias . . . 7
  - 1.3.2 Competencias no contempladas . . . 7
  - 1.3.3 Aplicaciones del Software . . . 7
- 1.4 Definiciones, acrónimos y abreviaturas . . . 8
  - 1.4.1 Definiciones . . . 8
  - 1.4.2 Acrónimos . . . 9
- 1.5 Visión General . . . 9

2 Descripción general del proyecto 11
- 2.1 Perspectiva del Proyecto . . . 11
  - 2.1.1 Interfaz del sistema . . . 11
  - 2.1.2 Interfaz de usuario . . . 12
  - 2.1.3 Interfaz software . . . 17
  - 2.1.4 Limitaciones de memoria . . . 17
  - 2.1.5 Operaciones . . . 17
  - 2.1.6 Requisitos generales . . . 18
- 2.2 Funciones del Producto . . . 18
- 2.3 Características del Usuario . . . 18
- 2.4 Restricciones Generales . . . 19
- 2.5 Suposiciones y Dependencias . . . 19
- 2.6 Desarrollo del calendario y presupuesto . . . 19
  - 2.6.1 División de tareas y asignación de tiempos y recursos . . . 19
- 2.7 Estimación de costes . . . 20

3 Desarrollo del proyecto 23
- 3.1 Introducción . . . 23
- 3.2 Metodología de desarrollo . . . 23

3.3 Especificación de requisitos del sistema . . . 24
3.3.1 Requisitos de información . . . 24
3.3.2 Requisitos funcionales . . . 25
3.3.3 Atributos del sistema Software . . . 25
3.3.4 Requisitos de rendimiento . . . 26
3.3.5 Restricciones de diseño . . . 26
3.4 Análisis del sistema . . . 26
3.4.1 Modelo de casos de uso . . . 26
3.4.2 Modelo conceptual de datos . . . 32
3.4.3 Modelo de comportamiento del sistema . . . 33
3.5 Diseño del sistema . . . 37
3.5.1 Arquitectura del sistema . . . 38
3.5.2 Diseño de la capa de gestión de datos . . . 39
3.6 Codificación . . . 42
3.6.1 Ingeniería inversa . . . 42
3.6.2 Almacenamiento y gestión de la información RDF . . . 43
3.6.3 Generación de grafos de los Linked Data . . . 44
3.6.4 Generador de RDFa . . . 44
3.6.5 Generación de URI's e interpretación . . . 45

4 Evaluación **47**
4.1 Pruebas y validación . . . 47
4.1.1 Pruebas de integración . . . 47
4.1.2 Pruebas de validación de RDF . . . 49
4.1.3 Pruebas de validación de RDFa . . . 50
4.2 Pruebas de la aplicación . . . 52
4.2.1 Integración de Redmine con Moodle . . . 52
4.2.2 Explotación de la información RDFa . . . 53
4.2.3 Indexación de información . . . 53

5 Conclusiones **55**
5.1 Conclusiones Generales . . . 55
5.2 Futuros Trabajos . . . 55

A Publicaciones **57**

B Manual de usuario **59**
B.1 Introducción . . . 59
B.2 Manual de Usuario: Administrador . . . 59
B.3 Manual de Usuario: Desarrollador . . . 61
B.4 Manual de Usuario: Usuario externo . . . 61

C Manual de instalación **63**
C.1 Requerimientos del sistema . . . 63
C.2 Instalación . . . 63
C.3 Configuración inicial . . . 64
C.4 Pruebas de instalación . . . 64

Bibliografía **65**

Índice de figuras

1.1 Ejemplo, elemento *perro* . . . 1
1.2 Ejemplo, grafo *Linked Data* basado en el concepto de *perro*. . . . 2
1.3 Gráfico de *Linked Data* . . . 8

2.1 Listado de la configuración de la información RDF asociada al modelo de datos. . . . 12
2.2 Configuración de la interfaz de EasyData. . . . 13
2.3 Edición de los datos de configuración de EasyData. . . . 14
2.4 Listado de la configuración de la información RDF asociada al modelo de datos. . . . 15
2.5 Listado de los datos de la aplicación que han sido publicados. . . . 16
2.6 Listado de los Linked Data de la aplicación. . . . 17

3.1 Casos de uso asociados al administrador . . . 27
3.2 Casos de uso asociados al usuario externo . . . 27
3.3 Casos de uso asociados al desarrollador . . . 28
3.4 UC-0201 Gestión de la información RDF asociada al modelo de datos . . . 28
3.5 UC-0501 y UC-0502 Alta/baja de los datos publicados. . . . 28
3.6 UC-0601 Gestión de la privacidad de los datos publicados. . . . 29
3.7 UC-0301 y UC-0302 Consulta de los datos publicados y la información sobre éstos. . . . 29
3.8 UC-0801 Integración de la información RDF asociada al modelo de datos en plantillas HTML (RDFa) . . . 29
3.9 Modelo conceptual de datos. . . . 32
3.10 Diagrama de Secuencia UC-0201. . . . 33
3.11 Diagrama de Secuencia UC-0501. . . . 33
3.12 Diagrama de Secuencia UC-0502. . . . 34
3.13 Diagrama de Secuencia UC-0601. . . . 34
3.14 Diagrama de Secuencia UC-0301. . . . 35
3.15 Diagrama de Secuencia UC-0302. . . . 35
3.16 Diagrama de Secuencia UC-0801. . . . 35
3.17 Arquitectura *software*. . . . 38
3.18 Función recolectora de modelos del proyecto anfitrión. . . . 42
3.19 Obtención de la información de los atributos de un modelo . . . 42
3.20 Obtención de la información de las relaciones de un modelo. . . . 43
3.21 Función generadora de los datos RDF iniciales. . . . 43
3.22 Acceso a la información almacenada en el fichero *rdf_info.yml*. . . . 44
3.23 Función generadora de los grafos Linked Data. . . . 44
3.24 Extensión de las rutas del proyecto anfitrión para contener las propias de la *gema*. . . . 46
3.25 Extracción de los parámetros de la consulta. . . . 46

4.1 Resultado de la prueba de validación de RDF. 50
4.2 Resultado de la prueba de validación de RDFa. 50
4.3 Redmine integrado en Moodle mediante un módulo que se sirve de EasyData. 52
4.4 Extensión de Firefox para la explotación de información RDFa. 53
4.5 Aplicación de indexación de información publicada con EasyData. 53

B.1 Captura de la pantalla de login 59
B.2 Captura de la pantalla de la edición de la información asociada al modelo de datos. 60
B.3 Captura de la pantalla de listado de la información publicada del modelo de datos. 61

Índice de cuadros

1.1 OBJ-01 Ingeniería inversa del modelo. 3
1.2 OBJ-02 Administración de la información RDF asociada al modelo de datos para su codificación. 3
1.3 OBJ-03 Interfaz de acceso a los *Linked Data* publicados mediante URI's desreferanciables 3
1.6 OBJ-06 Control de premisos de acceso a los datos publicados. 4
1.4 OBJ-04 Control de acceso a la interfaz de administración 4
1.5 OBJ-05 Control de la publicación de los datos del modelo de datos 4
1.7 OBJ-07 Publicación de la información de los datos publicados 4
1.8 OBJ-08 Generación de información RDFa 5

2.1 Listado de recursos utilizados en el desarrollo del proyecto. 20
2.2 Desglose de costes por recursos. 20

3.1 Requisito de información sobre modelos y atributos 24
3.2 Requisito de información sobre información RDF asociada al modelo de datos. 24
3.3 Requisito de información sobre el usuario administrador. 25
3.4 Entidades del sistema 39
3.5 Relaciones del sistema 39
3.6 Atributos de la entidad Modelo 40
3.7 Atributos de la entidad Dato 40
3.8 Atributos de la entidad Namespace 40
3.9 Atributos de la entidad Propiedad 41

1

Introducción

1.1. Introducción

Para entender el propósito de este proyecto es necesario conocer el contexto en el que surge la necesidad que viene a satisfacer.

La Web 2.0 actualmente la componen una cantidad muy extensa de aplicaciones en la red, las cuales, contienen una gran cantidad de datos almacenados de todo tipo. Si se publicasen y organizaran esta cantidad de datos, se podría generar una cantidad enorme de información disponible para cualquier usuario que la necesitase. Por tener un ejemplo, un investigador que busca la cura de un tipo de cáncer, con una herramienta que controlase y organizase éstos datos disponibles en la Web, podría obtener datos estadísticos acerca de las condiciones geográficas, medioambientales, poblacional, entre otras, que pueden marcar crucialmente su investigación y que costarían años y una importante inversión de recursos conseguir y que ahora estaría a tan sólo un *click*.Éste es un ejemplo de la potencia que la Web Semántica nos viene a ofrecer.

Con la Web Semántica surge un nuevo concepto, los Linked Data o datos vinculados. Los Linked Data vienen a expresar las conexiones entre los datos y cómo usando estas conexiones vamos a poder generar una gran cantidad de información útil y muy precisa. Para ilustrar qué son Linked Data, podemos verlo en el siguiente ejemplo:

- Tenemos un perro, el concepto de lo que es un perro.

Figura 1.1: *Ejemplo, elemento* perro

- ¿Qué define a un perro como concepto? Sus atributos, como por ejemplo: color, edad, tamaño, etc...También lo define datos como su raza, el cual también posee sus propios atributos, es decir, cuando pensamos en una raza de perro ya tenemos una visión de un tipo de perro, no nos imaginamos un caniche de 150 kilos ni un pastor alemán blanco, luego, la raza también tiene sus propios atributos. Aunque a un perro también

lo define si tiene dueño o no, y si lo tiene a éste lo definen sus atributos también.

Figura 1.2: *Ejemplo, grafo* Linked Data *basado en el concepto de* perro.

Podemos ver el grafo de datos vinculados que se ha formado que podría seguir desarrollándose. Así las búsquedas basadas en una estructura de datos son mucho más potentes, precisas y útiles que las realizadas por buscadores por contenidos nos ofrecen.

Bien, sabiendo los avances que supondrían tener aplicaciones que ofrecieran sus datos vinculados, ¿cuál es el problema? El problema radica en la necesidad de recursos que requiere actualizar las aplicaciones actuales en la Web y que, en la mayoría de los casos, sus propietarios no disponen de ellos o no ven la necesidad en invertir en éste propósito. En este punto es donde EasyData obtiene su valor y significado.

1.2. Objetivos

El objetivo principal del proyecto es la generación de un *plugin* o conocidos en términos de los desarrolladores del lenguaje Rudy, *gema*, cuya función es la generación automática de *Linked Data* del proyecto en el que es instalada. Para lograr este propósito, primero se deben salvar una serie de objetivos o subtareas que componen el objetivo principal. Estos objetivos o subtareas se listan a continuación:

OBJ-01	Ingeniería inversa del modelo de datos
Descripción	Ingeniería inversa del modelo de datos para obtener la lista de los modelos de la aplicación así como sus atributos.
Subobjetivos	–
Importancia	Muy Importante
Estabilidad	Alta
Comentarios	–

Cuadro 1.1: *OBJ-01 Ingeniería inversa del modelo.*

OBJ-02	Administración de la información RDF asociada al modelo de datos para su codificación.
Descripción	Asociación de las propiedades de los *namespace*s de RDF que la aplicación ofrece a los distintos atributos de cada modelo publicado.
Subobjetivos	–
Importancia	Muy Importante
Estabilidad	Alta
Comentarios	–

Cuadro 1.2: *OBJ-02 Administración de la información RDF asociada al modelo de datos para su codificación.*

OBJ-03	Interfaz de acceso a los *Linked Data* publicados mediante URI's desreferanciables
Descripción	Asociación de URI's desreferenciable a cada modelo de la aplicación para poder interpretar la información solicitada por el usuario externo.
Subobjetivos	–
Importancia	Muy Importante
Estabilidad	Alta
Comentarios	–

Cuadro 1.3: *OBJ-03 Interfaz de acceso a los* Linked Data *publicados mediante URI's desreferanciables*

OBJ-06	Control de premisos de acceso a los datos publicados
Descripción	Debe poder establecer un nivel de privacidad sobre los datos publicados en función del usuario que accede.
Subobjetivos	–
Importancia	Medio
Estabilidad	Alta
Comentarios	–

Cuadro 1.6: *OBJ-06 Control de premisos de acceso a los datos publicados.*

OBJ-04	Control de acceso a la interfaz de administración
Descripción	Control de acceso a la interfaz de administración.
Subobjetivos	–
Importancia	Importante
Estabilidad	Alta
Comentarios	–

Cuadro 1.4: *OBJ-04 Control de acceso a la interfaz de administración*

OBJ-05	Control de la publicación de los datos del modelo de datos
Descripción	La interfaz debe permitir elegir qué datos serán publicados y cuales no del modelo de datos del proyecto.
Subobjetivos	–
Importancia	Medio
Estabilidad	Alta
Comentarios	–

Cuadro 1.5: *OBJ-05 Control de la publicación de los datos del modelo de datos*

OBJ-07	Publicación de la información de los datos publicados
Descripción	Se debe mostrar la información acerca de qué se ha publicado y cómo se accede a estos datos.
Subobjetivos	–
Importancia	Medio
Estabilidad	Alta
Comentarios	–

Cuadro 1.7: *OBJ-07 Publicación de la información de los datos publicados*

OBJ-08	Generación de información RDFa
Descripción	Se desarrollará una herramienta que, a partir de la información RDF almacenada permita incrustar información RDFa en el código html del proyecto.
Subobjetivos	–
Importancia	Medio
Estabilidad	Alta
Comentarios	–

Cuadro 1.8: *OBJ-08 Generación de información RDFa*

A continuación, se detallan en más profundidad alguno de los objetivos clave del proyecto.

1.2.1. Ingeniería inversa del modelo de datos de la aplicación

Se trata de usar el modelo de datos de Ruby on Rails para generar un mapa de los modelos descritos en la aplicación. En esta parte, se asociarán URI's a los distintos recursos de la aplicación y, posteriormente, se publicarán para que los clientes puedan usarlas para obtener la información deseada. Este proceso implicará un reconocimiento de todos los modelos definidos así como de sus atributos. Esta tarea es el corazón del proyecto, ya que, establece las bases del acceso a la información de la aplicación y ofrece una herramienta a las entidades externas para consultarlos.

1.2.2. Administración de la información RDF asociada al modelo de datos para su codificación.

Dependiendo del modelo y los atributos en cuestión, la respuesta a la petición del usuario se elegirá el *namespace* de RDF necesario para la descripción de la información de respuesta.

Los *namespaces* de RDF (1.4.1) son listas de atributos orientadas cada una a un uso semántico y, por tanto, dependerá de la información expuesta en cada modelo la elección del *namespace* más útil en cada situación.

La lista de *namespaces* disponibles en el componente es la siguiente:

- Creative Commons (CC)
- Dublin Core (DC)
- Dublin Core 1.1 (DC11)
- Description of a Project (DOAP)
- Exchangeable Image File Format(EXIF)
- Friend of a friend (FOAF)
- Hyper transfer Protocol (HTTP)
- Web Ontology Lenguage (OWL)
- RDF Schema (RDFS)
- RDF Site Summary (RSS)
- SIOC

- Simple Knowledge Organization System (SKOS)
- Imported Web of Trust (WOT)
- Xhtml
- XSD built-in datatype property

Los *namespaces* no son más que unas reglas predefinidas para la generación del XML que generará como respuesta el sistema y que permitirá establecer un estándar para el mutuo acuerdo entre cliente y servidor de la información. De esta forma, se hace posible la interpretación de la información devuelta por ambas partes.

1.2.3. Control de acceso a la información mediante RBAC

No toda la información de la aplicación será accesible públicamente desde cualquier usuario o plataforma que desee consumir los *Linked Data* generados. Esto atributos no públicos pueden ser claves de acceso, información privada del usuario, cuentas de banco, información de empresas, etc... Este tipo de información no solo necesitará de un control para acceder a ellas sino que en algunos casos no son publicables bajo ningún concepto por distintos motivos: seguridad de la propia aplicación, vulnerabilidad de la ley de protección de datos, etc... Por tanto, el establecimiento de una capa de control gestionada por un RBAC (*Role-Based Access Control*) se hace fundamental para que el componente ha desarrollar sea usable en cualquier entorno de producción.

Mediante una interfaz de configuración del componente, entre otras posibilidades, se ofrecerá al desarrollador que realice la integración, la posibilidad de establecer los niveles de privacidad de los datos tanto a nivel de atributos como a nivel de modelos.

1.2.4. Interfaz de acceso a los *Linked Data* publicados mediante URI's desreferanciable

La generación de los *Linked Data* de la aplicación es un alto porcentaje del proyecto, pero no es útil si no hacemos saber al usuario que va a utilizar esta interfaz de comunicación los métodos para poder acceder a la información publicada. En esta tarea, se generará la estructura de publicación de uso de la interfaz de consumo de los *Linked Data* de la aplicación.

Se publicarán las URI's y los modelos asociados a ellas, así como, una visión general de la información publicada por la aplicación.

1.3. Alcance

Este proyecto se plantea resolver uno de los aspectos más importantes que la nueva Web solicita, y es la publicación y consumición de los datos almacenados en las distintas aplicaciones que se encuentran en ella. La Web Semántica propone un modelo de libre tránsito de datos para su consumo y su posterior transformación en información útil para distintas necesidades que se demanden. Esto es posible gracias a un sistema de publicación de dichos datos implantado en cada aplicación y que permita disponer sus datos a quien los solicite.

Este componente se centra, por tanto, en la generación de una interfaz de publicación de los datos almacenados por las aplicaciones desarrolladas bajo el *framework* Ruby on Rails. Encapsulada en una estructura denominada *gema*, este componente realiza una ingeniería inversa del modelo de datos de la aplicación y, posteriormente, genera una interfaz mediante la cual publica los datos de la aplicación para su consumo por terceros.

Por tanto, con este componente introduciremos a los desarrollos actuales y futuros realizados bajo Ruby on Rails en la Web Semántica.

El proyecto recibe el nombre de la *gema* resultante, EasyData, que representa la función principal del proyecto que es la publicación de los datos de una forma sencilla tanto para su integración como para su uso.

A continuación, se definirán las competencias de la *gema* EasyData y cuales no son sus competencias y que por razones de similitud de objetivos pueden llegar a pensar que las cumple también.

1.3.1. Competencias

Las competencias de la *gema* se basan en los objetivos descritos en la sección anterior. Estos objetivos son las distintas partes que necesita cumplir la *gema* para ofrecer la estructura de publicación de datos para la que será desarrollada. Ahora se listan las competencias que atañen a dicha *gema*:

- **Generación de un mapa del modelo de datos.** Este mapa servirá para poder generar la interfaz que trabajará para obtención de los datos solicitados.
- **Generación de una interfaz de comunicación.** Esta interfaz servirá como herramienta de consulta para quienes deseen consumir los datos almacenados en la aplicación.
- **Control de los datos publicados.** La generación automática sería un problema si no se basara en un sistema que controlase el acceso a determinados datos. Sin este control sería imposible no infringir la LOPD (Ley Orgánica de Protección de Datos), entre otros inconvenientes.

Estas son las competencias generales que abarca la *gema*. Para establecer bien los limites del desarrollo, a continuación se describirán alguna de las competencias que no abarcará la *gema* y que, por su semejanza a las que sí abarca, deben de indicarse para evitar su mal uso o su inclusión en entornos con necesidades distintas para las que se pretende desarrollar esta *gema*.

1.3.2. Competencias no contempladas

La Web Semántica requiere a las aplicaciones Web una serie de requisitos que deben cumplir. El libre tráfico de datos se produce en ambos sentidos, lo cual implica, no solo un control de los datos que se publican sino también un control (aun más exhaustivo) de los datos que se almacenarán por terceros. Este doble sentido de tráfico de datos para la aplicación no es una de las competencias de la *gema*, ya que, solo se centra en la publicación de datos y no en la generación de nuevos datos procedente de agentes externos.

1.3.3. Aplicaciones del Software

La diversidad de aplicaciones que puede tener el desarrollo de este componente para proyectos Ruby on Rails, se identifica directamente con los propósitos de la Web Semántica.

Por un lado, al disponer de los datos almacenados para consultas por terceros, permitirán construir cosechadoras de datos que consuman esta interfaz generada y permita la generación de un sistema de información como nunca antes hubo en la Web. Disponer de los datos de la aplicaciones de forma tan sencilla al resto de la Web permitirá la realización de estudios estadísticos con fines de investigación, construcción de buscadores semánticos, integraciones entre aplicaciones para ampliar los recursos ofrecidos a sus usuarios, entre otras grandes ventajas. Y no se acaba aquí, la libre circulación de datos por la Web establecido por

canales de transmisión de datos estándares como RDF o *Schema*, permitirá romper las barreras tecnológicas en la Web.

Tim Berners en su conferencia sobre Web Semántica en Ted Talks [16], expone las ventajas que supondría la Web 3.0 basada en la libre circulación de datos por parte de las aplicaciones que la componen. Mediante un gráfico 1.3 que podemos ver a continuación, vemos como utiliza la metáfora donde representa a los datos como semillas y la información como las plantas que nacen de éstas, simbolizando la principal ventaja de la publicación de los *Linked Data* de una aplicación y como este germen generaría una Web rica en información.

Figura 1.3: *Gráfico de* Linked Data

1.4. Definiciones, acrónimos y abreviaturas

1.4.1. Definiciones

Web Semántica o también denominada Web 3.0. La Web Semántica trata de una nueva forma de entender la Web que existe actualmente y cuyo mayor precursor es Tim Berners Lee padre del Hipertexto y, por tanto, de la Web 1.0. La Web Semántica pretende categorizar y etiquetar la inmensa cantidad de datos que esta en la Web con el objetivo de poder usarlo para construir una Web capaz de generar grandes cantidades de información muy útil y precisa.

Ruby. Lenguaje de programación dinámico. Creado por Yukihiro Matsumoto, mostró su primera versión en 1995. Este lenguaje posee grandes similitudes con otro lenguaje de programación llamado Perl. Se caracteriza por ser un lenguaje muy potente y amigable, lo cual, lo ha otorgado de una gran acogida en las comunidades de programadores de todo el mundo.

Ruby on Rails. Se trata de un framework basado en el lenguaje de programación Ruby. Creado por el danés David Heinemeier Hansson, dió sus primeros pasos en Diciembre de 2005. Ruby on Rails se basa en la idea de evitar la repetición innecesaria de código, uno de los grandes problemas de la programación actual.

Gema. Componente de Ruby instalable y usable en proyectos Ruby y, por lo cual, también en proyectos realizados con Ruby on Rails.

RDF *namespaces* contienen métodos y propiedades para definir y trabajar con espacios de nombre. Estos contienen las propiedades que luego se pueden asignar a cada atributo en el modelo de forma que describen el contenido de estos.

RDFa . Es un conjunto de extensiones de XHTML propuestas por W3C para introducir semántica en los documentos. RDFa aprovecha atributos de los elementos meta y link de XHTML y los generaliza de forma que puedan ser utilizados en otros elementos.

Linked Data. Los *Linked Data* o datos vinculados hacen referencia a esos datos que están relacionados de alguna forma entre sí. Este concepto surge con la Web Semántica, y viene a expresar la capacidad de mostrar, intercambiar y conectar datos a través de URI's desreferenciable en la Web.

1.4.2. Acrónimos

- **XML** Extended Markup Language
- **RoR** Ruby on Rails
- **RDF** Resource Description Framework
- **RDFa** Resource Description Framework–in–attributes
- **RBAC** Rol Base Access Control
- **DTO** Data Transfer Object
- **DAO** Data Access Object
- **LOPD** Ley Orgánica de Protección de Datos

1.5. Visión General

Este documento pretende recoger toda la fase de desarrollo del software EasyData, desde el planteamiento de las necesidades a las que se pretende dar solución, pasando por su planificación, su división en tareas, su distintas partes del desarrollo y, por último, la correspondiente guía de usuario para desarrolladores y clientes de los servicios ofrecidos.

En los siguientes puntos, se explicará la planificación del desarrollo software. Se expresará mediante un diagrama de *Gantt* la distribución de tiempos empleados en el desarrollo así como un informe de los gastos que supondría su realización con carácter comercial.

Seguidamente, se explicará el funcionamiento interno del software. Se realizará un exhaustivo análisis funcional del software desarrollado, abarcando desde sus casos de usos a la intervención de los actores con el software. Se expondrán también los requerimientos del sistema para la completa integración de la *gema* en el proyecto en cuestión.

Por último, y antes de generar el documento de usuarios, se expondrán los futuros desarrollos que se planifican realizar sobre el proyecto a modo de actualización y ampliación con los cuales se pretenderá ofrecer solución a un nuevo abanico de necesidades de la Web Semántica y de los proyectos que desean formar parte de ella.

2

Descripción general del proyecto

2.1. Perspectiva del Proyecto

EasyData es una gema perteneciente al listado disponible para los proyectos Ruby, y por transitividad, para los proyectos Ruby on Rails. Estas gemas son componentes *online* disponibles mediante la instalación del paquete *rubygems* y son desde funcionalidades que se añaden a los proyectos como: *captchar*, gestores de subidas de archivos, etc.. O pueden estar orientadas como herramientas de desarrollo del mismo proyecto. Este proyecto se encuentra entre ambos grupos, ya que se trata de un componente que facilita la generación de los Linked Data a los desarrolladores y, por otro lado, ofrece una interfaz de interoperabilidad para terceros que deseen integrarse con los datos almacenados en la aplicación. Los proyectos desarrollados bajo Ruby posee un repositorio *online* en el cual se alojan todos los componentes o gemas con los que podemos extender las funcionalidades de nuestros proyectos desarrollados bajo este *framework*. Si instalamos el paquete *rubygems* y posteriormente accedemos a la terminal y ejecutamos:

```
> gem list --remote
```

Se nos listará esta lista de componentes o gemas disponibles. Tras la finalización de este proyecto, EasyData se ofrecerá en esta lista a los desarrolladores para su instalación. Su funcionalidad es totalmente orientada a la Web. La actual base del proyecto es un componente del *framework* Ruby on Rails, lo cual, lo hace una solución específica para proyectos desarrollados con él. La idea en la que se fundamenta se puede exportar a otros lenguajes y *frameworks*, ya que, el método implementado no depende de la tecnología usada, que en éste caso es Ruby.

2.1.1. Interfaz del sistema

La interfaz del sistema en la que se basa el funcionamiento del proyecto se ve reflejada en la imagen de la arquitectura del sistema descrita en la sección 3.5.1. Podemos observar dos entornos de gemas, uno definido dentro del proyecto y otro fuera de él, en el entorno de Ruby del sistema. Esto se debe a que podemos instalar las gemas con distintos ámbitos. Las gemas que se localizan dentro de un proyecto se encuentran empaquetadas en él y lo hacen óptimo para su transporte en las distintas migraciones entre distintos servidores y entornos que pueda sufrir el proyecto a lo largo de su vida. En cambio, estas gemas sólo podrán usarse en el proyecto en cuestión. Por otro lado, las alojadas fuera del proyecto se vinculan al entorno Ruby del sistema en el que se encuentran los proyectos. De esta forma se pueden usar desde cualquier proyecto Ruby (y por tanto Ruby on Rails) alojado en el sistema. Si un proyecto hace uso de estas gemas deberá de asegurarse que existen instaladas en el sistema, o deberán empaquetarse al proyecto para asegurar su existencia en el entorno de despliegue.

2.1.2. Interfaz de usuario

Interfaz del usuario administrador

Esta es la interfaz orientada a la administración de la información RDF asociada a los modelos de la aplicación en la que se integra la gema EasyData. Presenta un formulario donde el usuario administrador podrá seleccionar las propiedades en base a los *namespaces* expuestos en este que se asociarán a cada atributo de cada modelo así como la privacidad y publicación de éstos. Esta interfaz orientada exclusivamente a administración será accesible mediante una identificación previa que comprobará que el usuario posee permisos de acceso previamente.

A continuación se muestran una serie de pantallas que servirán de guía para la generación de la interfaz de administrador.

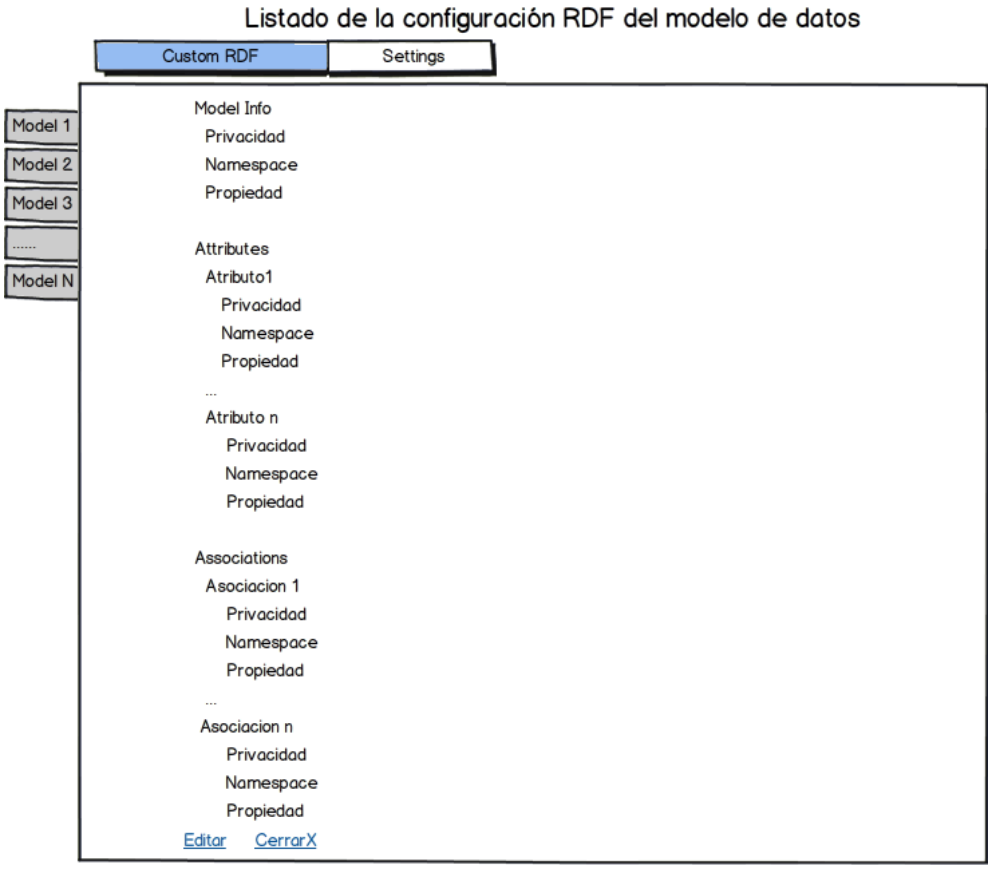


Figura 2.1: *Listado de la configuración de la información RDF asociada al modelo de datos.*

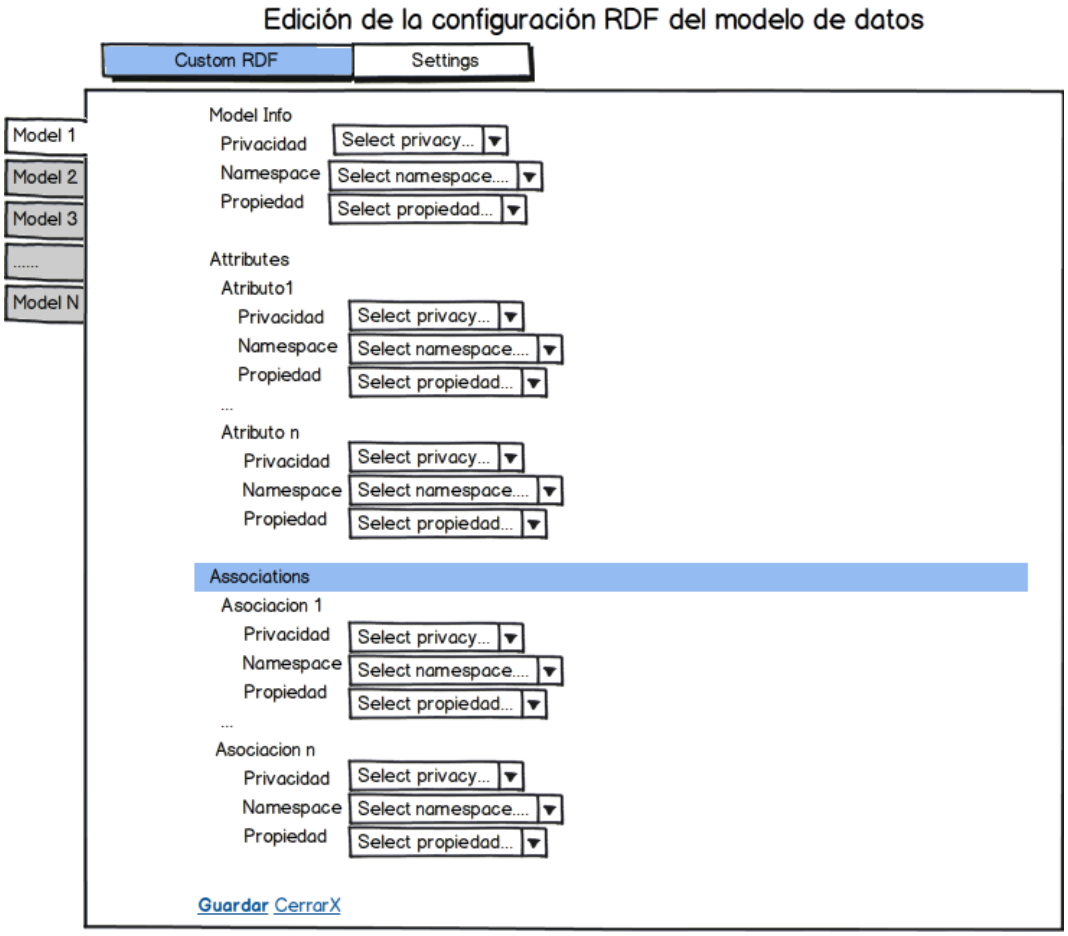


Figura 2.2: *Configuración de la interfaz de EasyData.*

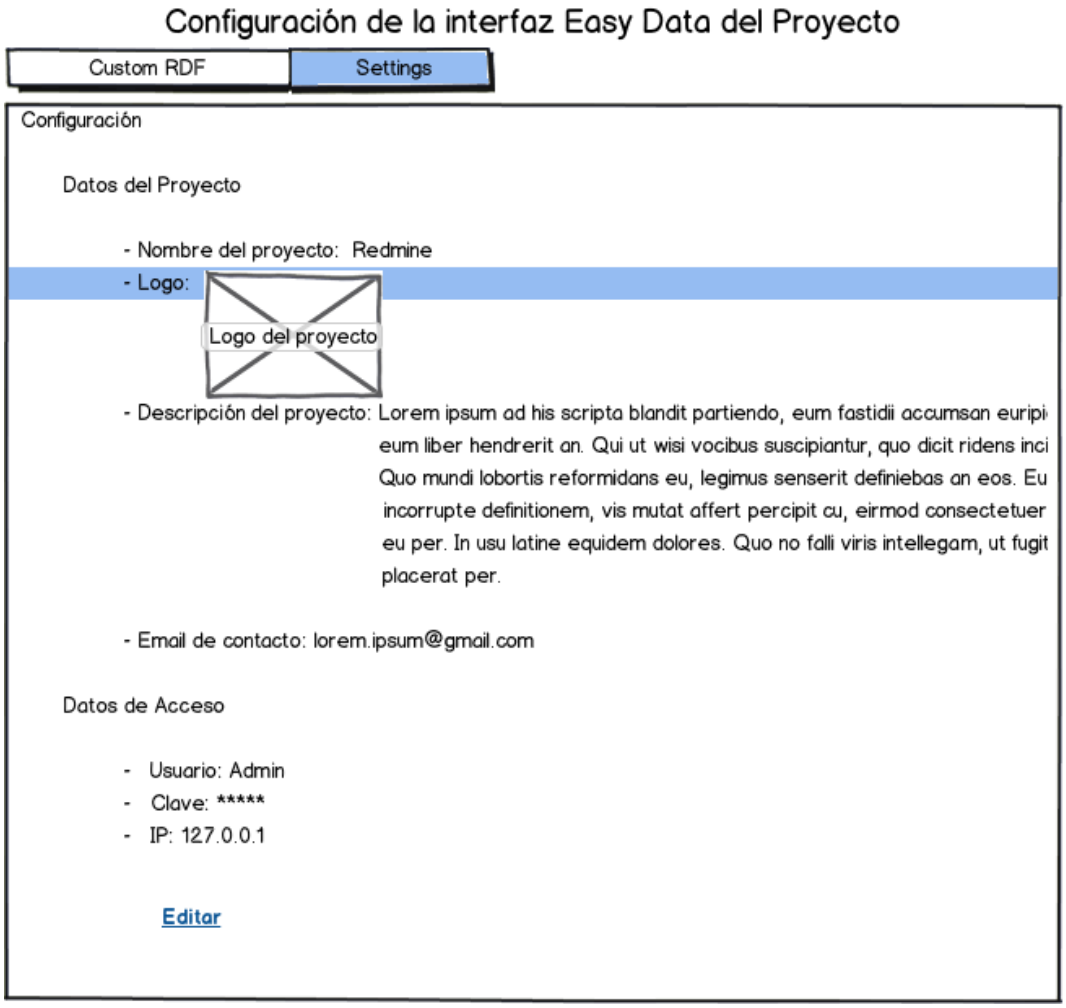


Figura 2.3: *Edición de los datos de configuración de EasyData.*

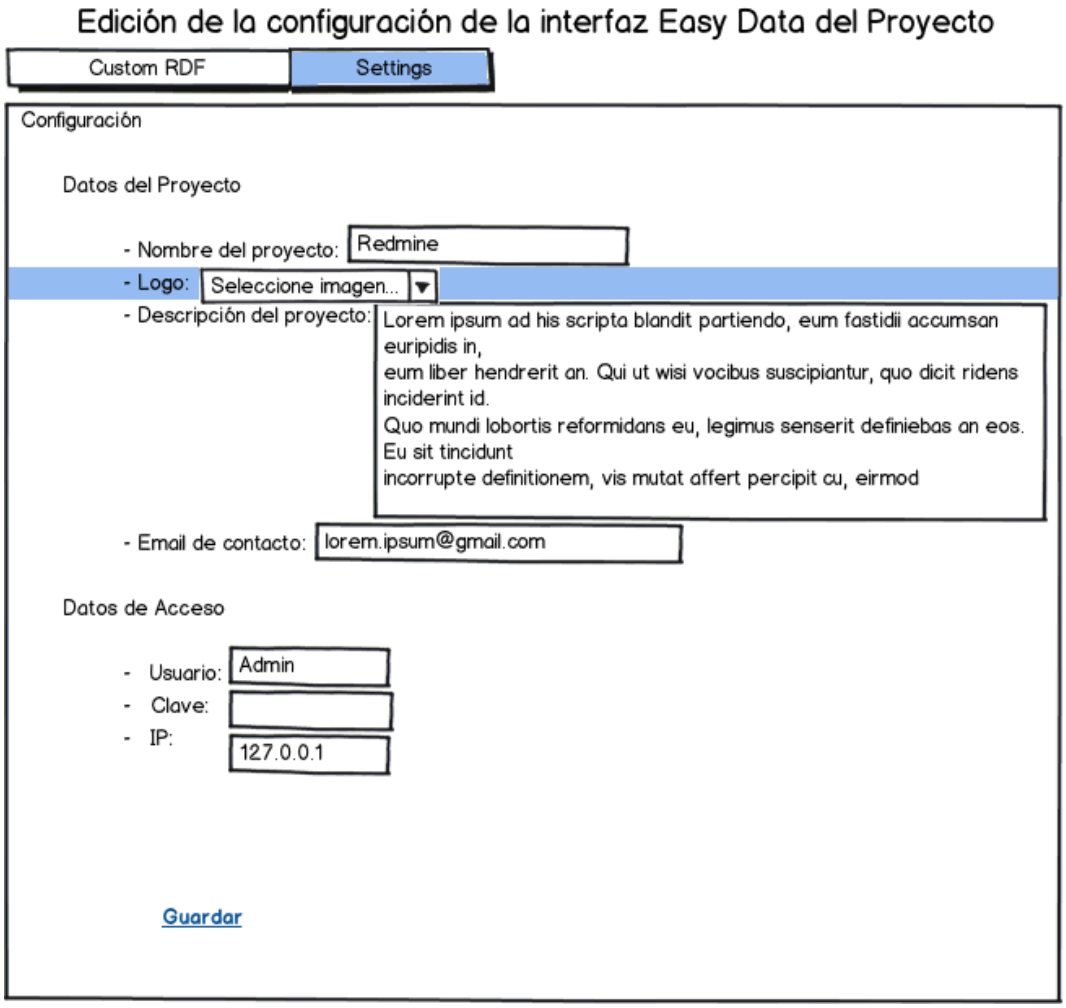


Figura 2.4: *Listado de la configuración de la información RDF asociada al modelo de datos.*

Usuario final

El usuario final hace referencia al usuario o plataforma que realizará las peticiones al sistema usando las herramientas que aporta EasyData a la aplicación. Este usuario podrá acceder a una interfaz donde se describe la información disponible para consultar así como un apartado donde se explica cómo se realizan las peticiones al sistema. A continuación, se muestran dos pantallas de la interfaz con el usuario final que ofrece EasyData. En éstas se aprecia en primera instancia, el listado de los datos publicados y en segunda, los datos vinculados que posee la aplicación.

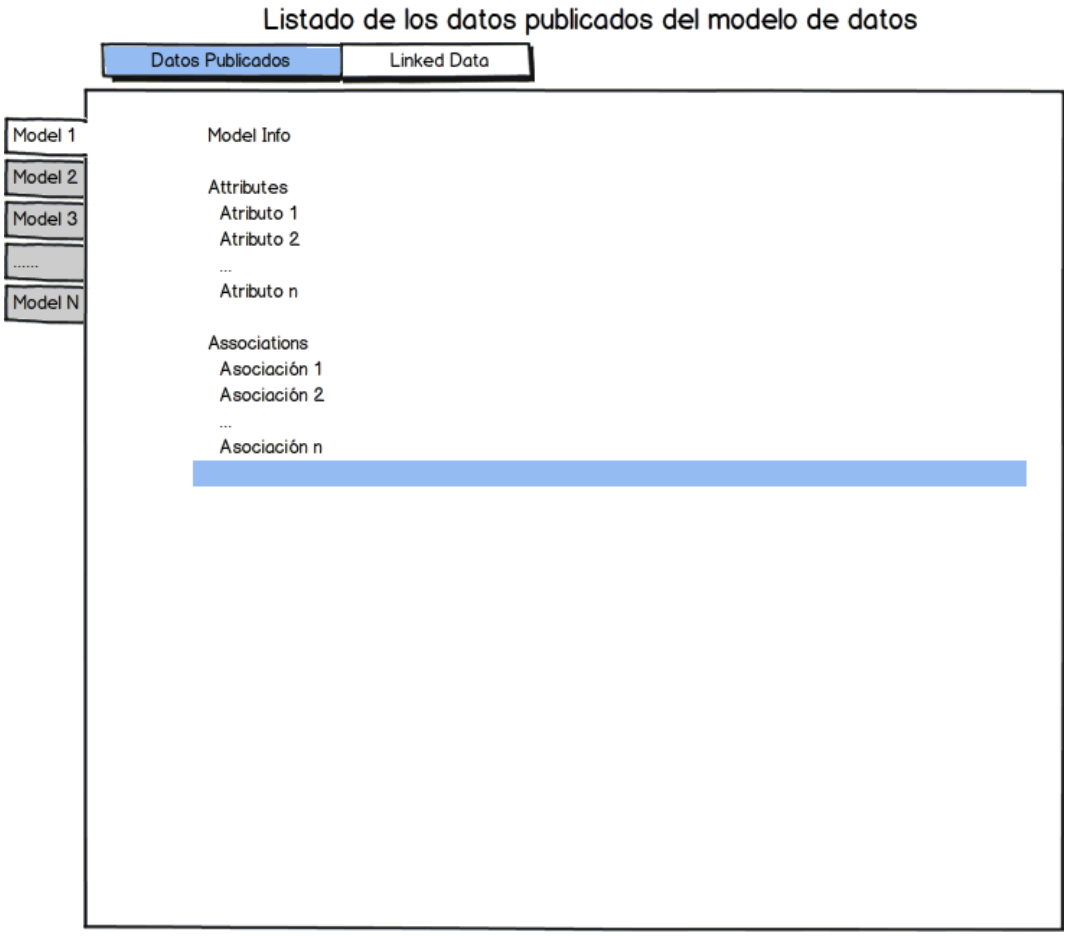


Figura 2.5: *Listado de los datos de la aplicación que han sido publicados.*

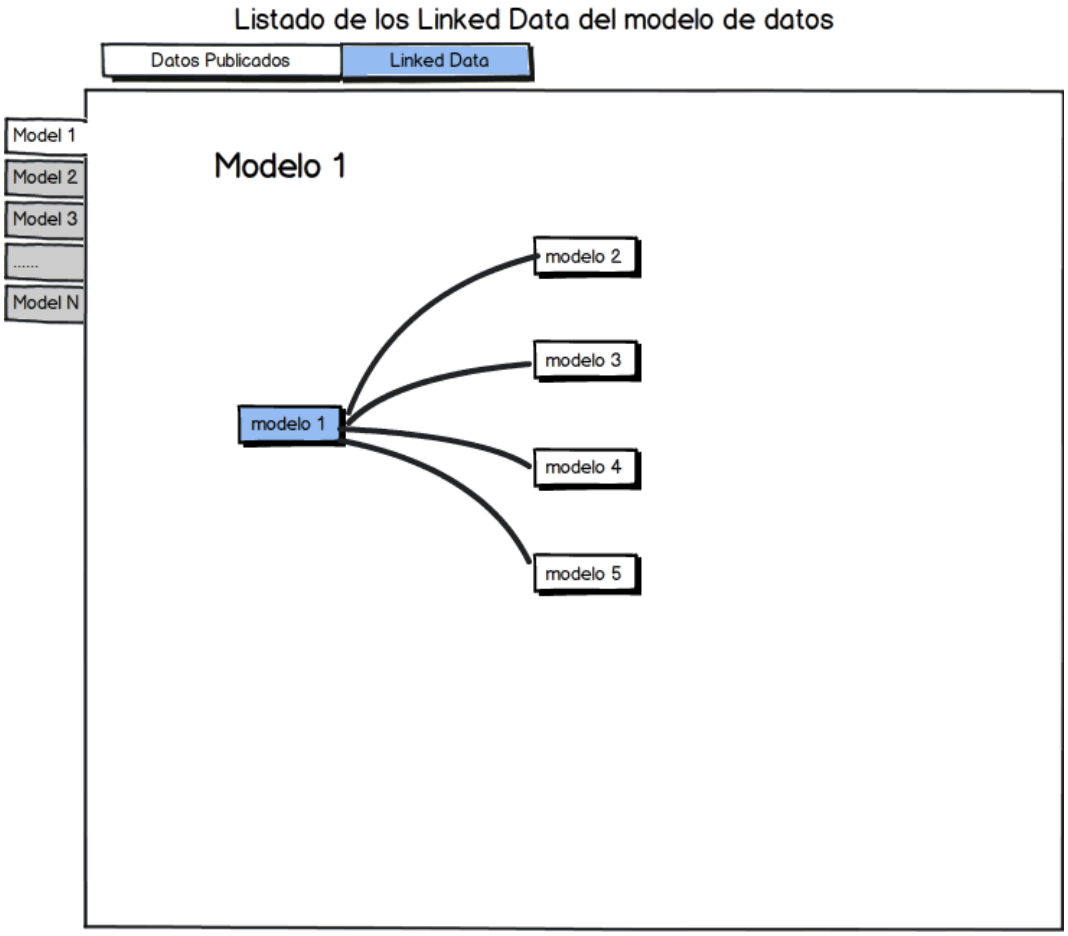


Figura 2.6: *Listado de los Linked Data de la aplicación.*

2.1.3. Interfaz software

El software está empaquetado en una gema. Esta gema es un componente que se encuentra alojado en un repositorio *online* accesible para todos los usuario sin requerimientos de autenticación, es de uso público. La gema se puede instalar en el entorno de Rails del servidor y ser accesible por todos los proyectos Rails alojados en él o puede ser empaquetada en un proyecto concreto y solo ser accesible por éste.

2.1.4. Limitaciones de memoria

Los requisitos de memoria de este *software* son muy limitados, ya que los únicos requisitos son el espacio en memoria necesario para poder alojar la gema y los documentos generados para la configuración del uso de ésta por el proyecto que la integra. Estos documentos son de texto plano y, por tanto, rara vez superarán los 100KB de memoria en su conjunto.

2.1.5. Operaciones

Operaciones de Administración

Las operaciones llevadas acabo por el administrador son aquellas mencionadas en el apartado de interfaz de usuario y hacen referencia a la administración de las propiedades

asociadas a los atributos de los modelos del sistema así como su publicación y nivel de privacidad.

Operaciones de Usuario

Las operaciones del usuario externo a la aplicación se resumen en la consulta de los datos publicados y el modo de acceso a éstos. Podrán visualizar el gráfo de Linked Data de la aplicación y ver las URI's asociadas a éstos, de forma que puedan realizar las peticiones de forma satisfactoria.

2.1.6. Requisitos generales

La gema esta desarrollada para un entorno con una versión de Ruby on Rails igual o superior a 2.3.5. Es el único requerimiento que posee este software.

2.2. Funciones del Producto

Gestión de la información RDF asociada al modelo de datos (Administrador). El sistema ofrecerá al usuario administrador una interfaz, mediante la cual, podrá relacionar cada atributo de cada modelo con una propiedad concreta de un determinado namespace de RDF.

Generación automática de una interfaz de acceso al modelo mediante URI's desreferenciable. El proyecto generará una lectura de los modelos definidos en el proyecto Ruby on Rails y un listado de URI's desreferenciables mediante las cuales se podrá acceder a los datos que éstos gestionan.

Gestión de los niveles de privacidad de los datos publicados (Administrador). Mediante la interfaz de administración, el usuario administrador podrá decidir el nivel de privacidad necesario que debe tener el usuario que realice una petición para poder acceder a un determinado atributo del modelo. De esta forma se controla que no todos los datos del proyecto sean de acceso público.

Gestión de la publicación de los datos contenidos en el modelo de datos. La interfaz además permitirá decidir sobre qué atributos se mostrarán en las solicitudes y cuales no pudiéndose ocultar a las peticiones de los usuarios.

Consulta de los datos publicados (usuario externo). Cada acceso por URI's generará una respuesta con los datos solicitados en formato XML siguiendo la estructura RDF configurada por el administrador.

Consulta del modo de acceso a los datos publicados (usuario externo). La gema generará una interfaz de acceso público donde se mostrará la informa de acceso a los datos publicados. Se indicarán la relación de URI's y los modelos a los que referencia así como la interconexión entre estos.

2.3. Características del Usuario

Este proyecto va orientado a generar o mejorar la interoperabilidad de las aplicaciones desarrolladas bajo Ruby on Rails. Esta interoperabilidad se realiza entre desarrolladores, por un lado, los encargados de gestionar la interfaz de publicación de datos del proyecto y, por otro lado, los que consumen la información publicada. Por tanto, el usuario de la gema tanto en integración como en consumo de los Linked Data generados son de carácter desarrollador y/o administrador.

El usuario encargado de la integración de EasyData en la aplicación, debe ser un usuario con conocimientos de Ruby on Rails. Debe conocer la técnica de instalación de gemas y la ejecución de tareas de configuración que requiere tras la instalación en el sistema. A continuación, para las tareas de administración y configuración de la información de RDF asociada a los modelos, el usuario debe tener conocimientos de RDF a un nivel avanzado para poder realizar la configuración más óptima para el modelo de datos de la aplicación.

El usuario que va a consumir la información publicada debe conocer las técnicas de interpretación de XML en el lenguaje que precise la aplicación donde se va realizar la integración. Debe tener conocimientos de uso de URI's para poder realizar las consultas de forma correcta y no obtener resultados erróneos o no deseados.

Este proyecto tiene un carácter muy técnico y, como tal, sus usuarios deben tener tales conocimientos, por tanto, los usuarios de software serán en cualquier caso administradores y/o desarrolladores.

2.4. Restricciones Generales

La principal restricción del software es que debe usarse Ruby como lenguaje de programación. Debido a que es un software que se integra con proyectos desarrollados en Ruby on Rails, esta restricción es trivial.
Por otro lado, al tratarse de un sistema semiautomático (por requerir de una configuración inicial por parte del usuario) debe tenerse en cuenta que la publicación de ciertos datos puede violar la LOPD, por ello, se deberá ofrecer la posibilidad de no publicar o restringir el acceso a este tipo de datos.
No como restricción sino como consejo, se debe seguir el paradigma de la programación orientada a objetos para un mantenimiento más eficaz y una mejor integración, ya que, el sistema anfitrión (Ruby on Rails) sigue dicho paradigma.

2.5. Suposiciones y Dependencias

Este software depende en su funcionamiento de otras bibliotecas o componentes que facilitan algunas tareas que debe llevar a cabo. A continuación, se listan las dependencias del sistema en cuanto a biblioteca, para la integración de la gema EasyData con los distintos proyectos.

YAML librería de renderización e interpretación de archivos *.yaml

Yard gema para la generación de documentación sobre el código, basándose en los comentarios de éste.

Rake gema para la ejecutar las tareas programadas para la instalación y configuración inicial de la gema en el proyecto.

2.6. Desarrollo del calendario y presupuesto

2.6.1. División de tareas y asignación de tiempos y recursos

En este apartado se describe la división de tareas realizada para planificar y llevar a cabo el proyecto EasyData. El proyecto se dividió en tareas que se agruparon en las distintas etapas del desarrollo software: investigación, análisis y planificación, desarrollo y testeo del producto software. El proceso de documentación del proyecto empezó una vez realizada el 50 % del desarrollo del software y se realizó de forma paralela a las tareas de desarrollo que se realizaban en el momento.

A continuación se van a mostrar el gráficos que recoge el diagrama de Gantt del proyecto donde podemos observar el tiempo y la secuencia de las tareas que comprende el desarrollo del proyecto.

2.7. Estimación de costes

Para realizar una estimación de los costes del proyecto es necesario evaluar todos los recursos implicados en el desarrollo de éste. A continuación se van a listar los recursos humanos y herramientas que han sido necesarias para llevar a cabo el proyecto EasyData. Hay que tener en cuenta que el material de oficina que se va a listar, aunque sufre un deterioro normal por su uso, no es apto para tener en cuenta en el presupuesto debido a que cualquier empresa *software* debe disponer entre sus activos de elementos necesarios para desarrollar proyectos como éste y no suponen una inversión para dicho propósito. También se debe tener en cuenta aquellos gastos indirectos propios de las empresas del sector *software* como lo son la conexión a internet, material de oficina, recambio de impresoras, etc.. Estos elementos serán englobados bajo el término de costes indirectos y su coste suele ser entre un 10 % y un 20 % del coste del personal, para este caso, supondremos un coste medio del 15 % del coste del personal. Otros costes como lo son los de tipo inventario, en este caso la computadora, tienen un período de amortización normalmente estipulado entre 2 y 4 años. Para este caso, supondremos un período de amortización de 3 años que, por tanto, si su coste asciende a 1200 euros amortizaremos 400 euros al año de dicho inventario y, como el período de desarrollo es de 6 meses, el coste final será de 200 euros el coste de dicho equipamiento.

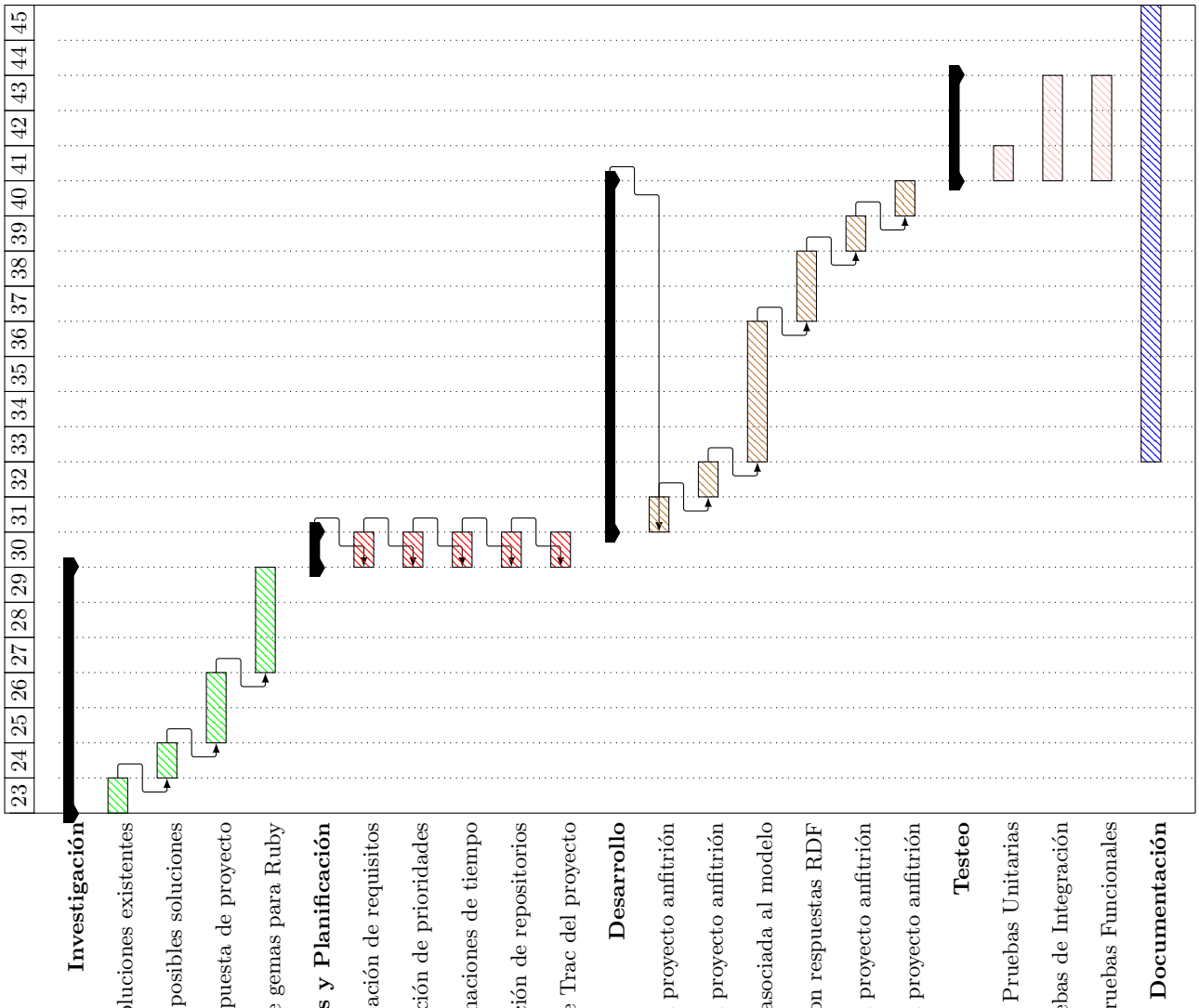
Nombre	**Descripción**	**Tipo**	**Inversión estimada**
Desarrollador	Personal cualificado y especializado	Humano	26.000 euros/año
Computadora	Puesto de trabajo.	Activo	200 euros
Costes indirectos	Bolígrafos, papel, tinta, etc ...	—	15 % del coste del personal.

Cuadro 2.1: *Listado de recursos utilizados en el desarrollo del proyecto.*

En la tabla podemos apreciar los distintos recursos necesarios para llevar a cabo el desarrollo de éste proyecto. Para estimar el coste del programador, se ha consultado la media salarial en Infojobs que lo establece en 26.000 euros anuales brutos a los que hay que sumarles los costes de seguridad social que, por regla general, son una tercera parte del empleado, es decir, 618,5 euros mensuales. A continuación, se procederá a establecer el desglose de costes asociados a los recursos en el período de desarrollo del proyecto EasyData. Se ha de tener en cuenta que los activos de la empresa, por regla general, ya los poseía y no requerirán de una inversión. Aun así, se van a considerar dentro de la estimación de costes.

Cantidad	**Descripción**	**coste unitario**	**coste total**
1	Desarrollador	2.470 euros/mes	12.350 euros
1	Estación de trabajo	200 euros	200 euros
*	costes indirectos	15 % coste del personal	370 euros
Total			12.920 euros

Cuadro 2.2: *Desglose de costes por recursos.*


23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
Investigación
Búsqueda de soluciones existentes
Investigación de posibles soluciones
Propuesta de proyecto
Desarrollo de gemas para Ruby
Análisis y Planificación
Identificación de requisitos
Asignación de prioridades
Estimaciones de tiempo
Generación de repositorios
Generación de Trac del proyecto
Desarrollo
Integración de funcionalidades de la gema con el proyecto anfitrión
Captura de información del modelo de datos del proyecto anfitrión
Intefaz de gestión de información RDF asociada al modelo
Generación de la interfaz REST con respuestas RDF
Generador de información RDFa para la capa Vista del proyecto anfitrión
Interfaz de información de la publicación de datos del proyecto anfitrión
Testeo
Pruebas Unitarias
Pruebas de Integración
Pruebas Funcionales
Documentación

El desglose de costes asociados al desarrollador se ha realizado en base a los siguientes datos:

- Salario mensual. Estipulando que el sueldo anual indicado por *Infojob* es de media 26.000 euros brutos anuales, repartidos en 14 pagas obtenemos los 1857,15 euros al mes.
- El 33 % del sueldo mensual suele ser la cantidad a pagar de seguridad social, en este caso asciende a 612.85 euros.

El total mensual de los costes asociados al desarrollador ascienden a 2.470 euros.

Como se indicó al principio de esta sección, los costes asociados a los activos son una inversión que hipotéticamente una empresa dedicada al desarrollo *software* no tendría que realizar por disponer de este activo implícito para su actividad diaria. Pero se ha incluido para estimar un coste que cubre las posibles necesidades que puedan surgir durante el tiempo de desarrollo, como por ejemplo, que la estación de trabajo sufra una avería sin solución y necesitase adquirir una nueva.

3

Desarrollo del proyecto

3.1. Introducción

Este proyecto pertenece a la modalidad de Proyectos de Desarrollo *software*. Aunque no se trata de un producto *software* con autonomía, ya que, requiere de un sistema *software* anfitrión como lo es Ruby on Rails y para el que está orientado este *software*.

A continuación, se van a explicar en las siguientes secciones los distintos aspectos que comprenden el desarrollo de este producto para dar a ver de manera más técnica su desarrollo.

3.2. Metodología de desarrollo

La metodología de desarrollo seguida se basa en el *Proceso Unificado de Rational* (RUP), basado a su vez en el modelo de desarrollo en espiral. Utilizar la metodología descrita por RUP permitirá dividir el proyecto *software* en distintas etapas que facilitarán la organización de su desarrollo, estableciendo unos puntos de evaluación de las evoluciones del proyecto que nos permitirá ir avanzando en su desarrollo generando versiones estables a su paso. El desarrollo en espiral se compone de distintas fases que comprenden una serie de etapas que la definen. Éstas etapas en este proyecto son las siguiente:

1. **Análisis del problema**. En cada etapa se abordará un nuevo objetivo en los que se divide el proyecto. Dicho objetivo es estudiado y analizado para seleccionar las herramientas y métodos adecuados para su resolución. Se analizan sus requisitos y esto permitirá tener un concepto a priori del desarrollo de la solución y de los alcances que va a tener ésta.

2. **Planificación**. Una vez identificado y analizado el problema a resolver, se divide el problema en tareas, subtareas y se asignan los tiempos de desarrollo con un margen de error para no vernos demasiado comprometidos con éstos. A continuación se estiman los recursos que serán necesarios para llevar a cabo el desarrollo de cada tarea y subtarea. De esta forma, tenemos una visión más veraz del tiempo y recursos que implicarán el desarrollo de la solución.

3. **Desarrollo**. Tras analizar y planificar, ya estamos en condiciones de empezar con el desarrollo de la solución del objetivo en cuestión. Se requiere de cumplir el desarrollo de las tareas y subtareas en los tiempos estipulados para no ver perjudicados el resto de los objetivos en los que se divide el proyecto en tiempo y recurso.

4. **Evaluación**. Tras el desarrollo de la solución, debemos comprobar que se ha obtenido una solución real al problema propuesto y que no obtenemos ningún resultado erróneo o inesperado.

Tras la finalización del proceso de evaluación de cada fase se vuelve a el análisis con un nuevo objetivo del proyecto, de esta forma se va avanzando y generando *software* estable y útil que finalmente supondrá la solución al problema que pretende dar solución este proyecto.

Para diseñar el modelo de datos y su documentación se va a utilizar el Lenguaje Unificado de Modelado (UML) y, por su perfecta sincronización, el paradigma de programación Orientado a Objeto (POO). El uso de éste paradigma de programación no sólo se ha decidido por su sincronía con UML, sino orientado a una perfecta sincronización con el sistema anfitrión que también se basa en él y en el patrón modelo-vista-controlador (MVC) del que también hace uso éste *software*. Esta elección permitirá además de una mejor integración, una mayor facilidad de su mantenimiento futuro.

3.3. Especificación de requisitos del sistema

A continuación, se especificarán los distintos requisitos que presenta el proyecto.

3.3.1. Requisitos de información

Los requisitos de información nos expresarán los distintos grupos de información requeridos por el sistema para llevar a cabo los objetivos con los que está relacionado. Los requisitos de información del proyecto son los siguientes:

IRQ-01	Modelos y atributos
Objetivos asociados	OBJ-01 Reconocimientos del modelo de datos
Requisitos asociados	–
Datos Específicos	Modelo Atributos
Estabilidad	Alta
Comentarios	–

Cuadro 3.1: *Requisito de información sobre modelos y atributos*

IRQ-02	Información RDF asociada al modelo de datos
Objetivos asociados	OBJ-01 Ingeniería inversa del Modelo de Datos OBJ-02 Administración de la información RDF asociada al modelo de datos
Requisitos asociados	IRQ-01 Modelos y Atributos
Datos Específicos	Modelo Atributo Namespace Propiedad
Estabilidad	Alta
Comentarios	–

Cuadro 3.2: *Requisito de información sobre información RDF asociada al modelo de datos.*

IRQ-03	Usuario Administrador
Objetivos asociados	OBJ-04 Control de acceso a la interfaz de administración
Requisitos asociados	–
Datos Específicos	Usuario Clave
Estabilidad	Alta
Comentarios	–

Cuadro 3.3: *Requisito de información sobre el usuario administrador.*

3.3.2. Requisitos funcionales

A continuación, se detallan los requisitos funcionales que debe cumplir el proyecto software.

- **Administrador**
  - Administrar la información RDF asociada al modelo de datos.
  - Publicación y des-publicación de datos del modelo.
  - Establecer privacidad de los datos publicados del modelo.
  - Visualización de los datos publicados.
- **Usuario Externo**
  - Acceso a los datos publicados mediante URI's desreferenciables.
  - Visualización de los datos publicados.

Estos requisitos se explicarán con más detalle en el apartado de casos de usos, donde se expondrán cada uno de éstos junto con los actores que intervienen en ellos.

3.3.3. Atributos del sistema Software

El sistema debe cumplir una serie de requisitos que son independiente de los objetivos principales del proyecto. Estos requisitos actuarán como índices de calidad del producto software desarrollado y darán valor a éste frente a los usuarios.

- **Disponibilidad**. Al tratarse de un componente de un sistema software, este debe estar disponible para su instalación por parte de los desarrolladores. Para ello el sistema debe estar accesible para su instalación siempre que el usuario desarrollador lo solicite. Por ello, la gema EasyData se alojará en el repositorio público `www.RubyGem.org`.
- **Fiabilidad**. El sistema debe ser fiable en cuanto a los datos que ofrece a los usuario externos como a la información de configuración que muestra al usuario administrador.
- **Seguridad**. Debe ofrecer un sistema de protección de los datos que no se desean que sean de acceso público. Debe ser seguro el sistema de configuración de información de RDF asociado al modelo, ya que, puede verse comprometida la exposición incorrecta de los datos contenidos por el modelo de datos de la aplicación.
- **Mantenibilidad**. El sistema debe ser mantenible. Su existencia esta ligada al Ruby on Rails y, por tanto, debe ser fácil su actualización para poder seguir siendo funcional en las siguientes versiones del framework.

- **Portabilidad**. Debe ser instalable e integrable en cualquier proyecto que funcione bajo el framework Ruby on Rails.

Estos requisitos deben ser cumplidos por el software independientemente que cumpla los objetivos principales establecidos.

3.3.4. Requisitos de rendimiento

Los requerimientos del sistema en cuanto a rendimiento se centran en la capacidad de generar una respuesta a la petición del usuario externo en un tiempo razonable. Este punto es un clave para que sea una herramienta útil y factible de integrar en los proyectos desarrollados usando Ruby on Rails. Pero al tratarse de un complemento añadido a un proyecto software anfitrión, los recursos de los que se disponga éste y la calidad con la que se ha desarrollado serán factores influyentes en el rendimiento de las funcionalidades de EasyData. Por tanto, se debe tener en cuenta que la respuesta a las solicitudes, en las pruebas realizadas, pueden verse influidas por elementos externo a este software.

3.3.5. Restricciones de diseño

La funciones principales del software no incluyen un diseño visual, salvo los relacionados con la administración y configuración de la información RDF. Este parte del proyecto debe ser diseñada atendiendo a las reglas establecidas por HTML 4.x y CSS 2.x. Su simplicidad inicial debe ser lo más neutral posible, ya que, no es una herramienta que podrán ver los usuarios finales de la aplicación a la que sea integrada. Debe ofrecer una hoja de estilos modificable por los desarrolladores y maquetadores que deseen otorgar una apariencia similar al resto del proyecto software en cuestión. Atendiendo a las respuestas a las solicitudes de los usuarios externos que consultan los datos publicados, la respuesta será codificada en formato XML 1.0 y siguiendo las reglas establecidas por los distintos *namespaces* de RDF usados.

3.4. Análisis del sistema

Para el análisis del sistema se utilizará un enfoque orientado a objetos que permitirá una mayor facilidad y veracidad en su cometido, ya que el proyecto estará también desarrollado usando el paradigma de la orientación a objetos.

3.4.1. Modelo de casos de uso

Definición de los actores del sistema

- **Administrador**. Esta figura representa a la persona encargada de integrar la *gema* en el sistema. Su ratio de acción será la de indicar la información necesaria para la puesta en marcha del componente dentro del proyecto en el que es integrada. Esta persona es la de mayor responsabilidad en el sistema, ya que decidirá no solo sobre que datos RDF se asocian a los modelos si no que establecerá la privacidad de los datos publicados así como tomará la decisión de cuales se publican y cuales no.

- **Desarrollador**. Este actor cobra importancia en un solo caso de uso y representa al desarrollador de la aplicación *software* anfitriona de la *gema*.

- **Usuario Externo**. Este actor representa no solo a personas físicas, sino también a otros sistemas que realizan un uso de este componente para consultar los datos gestionados por el modelo de datos del proyecto en cuestión. Su responsabilidad se centra en la correcta interpretación de las respuestas así como la correcta utilización del sistema de consultas mediante URI's desreferenciables.

Diagramas y especificación de casos de uso

A continuación se van a indicar todos los casos de uso del sistema asociados a los actores que intervienen en cada uno de ellos. Esta sección, complementa la definición anterior de los requisitos funcionales del sistema. Los dos primeros dígitos de la numeración de los casos de uso indica el objetivo al que está asociado, mientras que los dos últimos dígitos quedan reservados para numerar al propio caso de uso. Para las solicitudes de los usuarios externos, la información se verá afectada dependiendo de si éste está autentificado o no, pero no es un requisito previo a ningún caso de uso relacionado con las consultas.

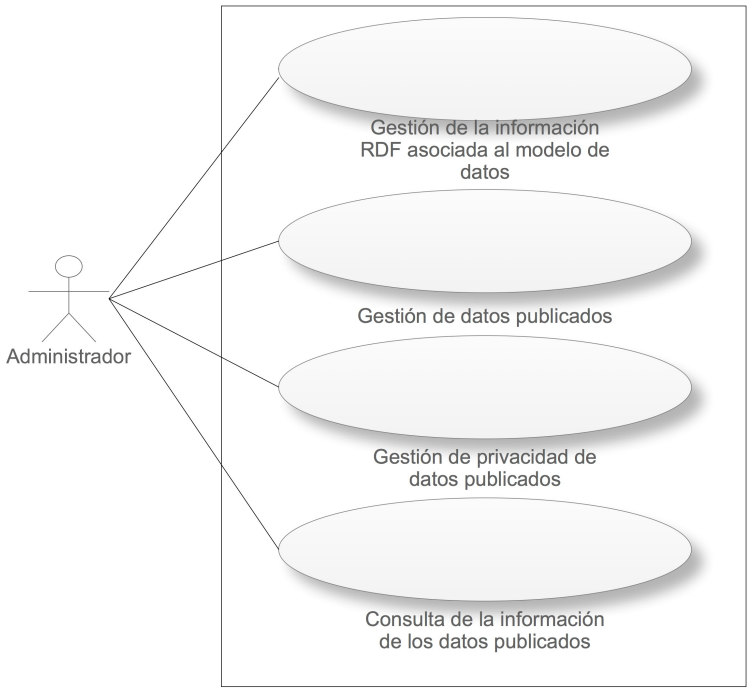


Figura 3.1: *Casos de uso asociados al administrador*

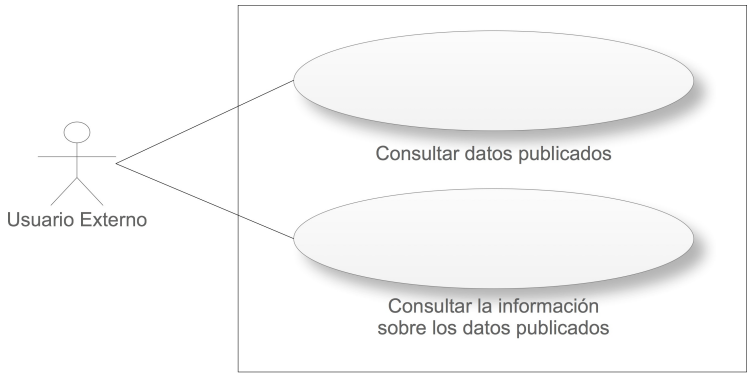


Figura 3.2: *Casos de uso asociados al usuario externo*

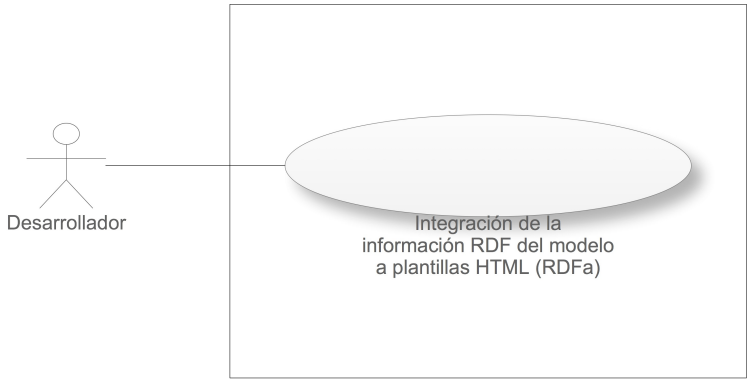


Figura 3.3: *Casos de uso asociados al desarrollador*

Ahora se listan los desgloses de los casos de usos compuestos en los casos de usos simples que lo componen.

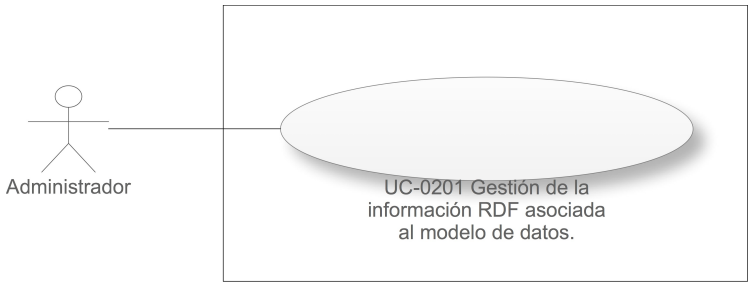


Figura 3.4: *UC-0201 Gestión de la información RDF asociada al modelo de datos*

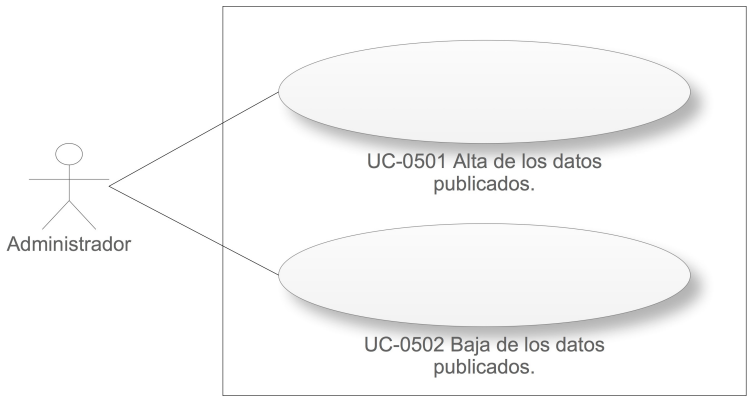


Figura 3.5: *UC-0501 y UC-0502 Alta/baja de los datos publicados.*

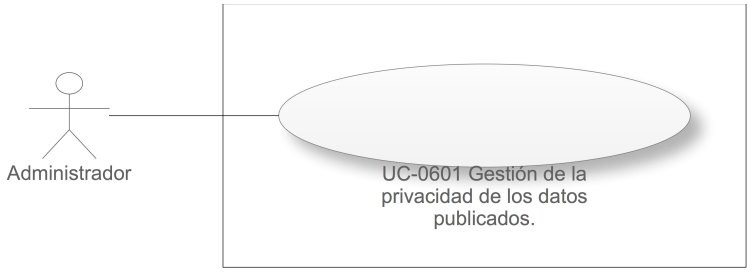


Figura 3.6: *UC-0601 Gestión de la privacidad de los datos publicados.*

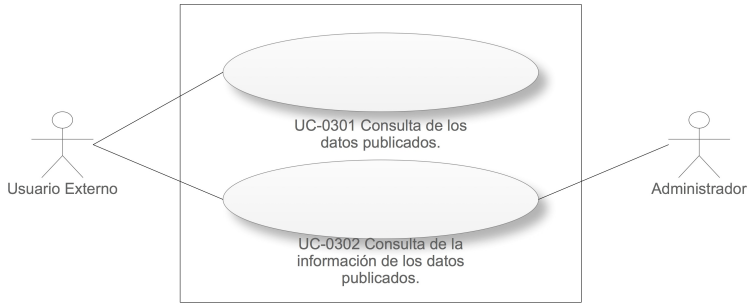


Figura 3.7: *UC-0301 y UC-0302 Consulta de los datos publicados y la información sobre éstos.*

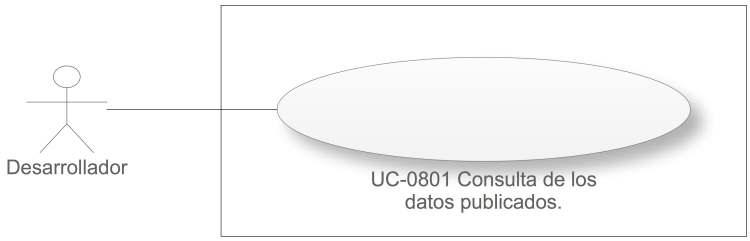


Figura 3.8: *UC-0801 Integración de la información RDF asociada al modelo de datos en plantillas HTML (RDFa)*

- **UC-0201 Gestión de la información RDF asociada al modelo de datos**
  - **Descripción**: El administrador asocia propiedades de los *namespaces* listados a los atributos de cada modelo.
  - **Precondición**: Se debe haber generado la información de realizar la ingeniería inversa del modelo de datos para poder listarse en el panel de administración.
  - **Actores**: administrador
  - **Escenario principal**

    1. El administrador accede mediante su identificación al panel de gestión de información RDF del modelo de datos.
    2. Selecciona un modelo de la lista de éstos.
    3. Pincha en el botón de modo de edición.
    4. Elige un atributo para añadir o editar su información RDF asociada.
    5. Selecciona el *namespace* y, a continuación, la propiedad de éste que desea asociar al atributo.

6. Se repiten los pasos 4 y 5 hasta que finalice la tarea de gestión de información de RDF sobre el modelo en concreto, y pincha en enviar.

7. El sistema vuelve a mostrar el listado de atributos con su información RDF asociada en modo no edición.

- **UC-0501 Alta de los datos publicados del modelo de datos**
  - **Descripción**: El administrador da de alta un atributo que aparecía como oculto a las consultas del usuario externo.
  - **Precondición**: Se debe haber generado la información de realizar la ingeniería inversa del modelo de datos para poder listarse en el panel de administración.
  - **Actores**: administrador
  - **Escenario principal**:

    1. El administrador accede mediante su identificación al panel de gestión de información RDF del modelo de datos.

    2. Selecciona un modelo de la lista de éstos.

    3. Pincha en el botón de modo de edición.

    4. Elige un atributo y le asocia un estado de privacidad distinto de "oculto".

    5. El administrador repite el paso anterior hasta que termina de publicar atributos del modelo y pincha en el botón "Enviar" .

    6. El sistema vuelve a mostrar el listado de atributos con su información RDF asociada en modo no edición.

- **UC-0502 Baja de los datos publicados del modelo de datos**
  - **Descripción**: El administrador da de baja un atributo que aparecía como público o privado a las consultas del usuario externo.
  - **Precondición**: Se debe haber generado la información de realizar la ingeniería inversa del modelo de datos, para poder listarse en el panel de administración.
  - **Actores**: administrador
  - **Escenario principal**:

    1. El administrador accede mediante su identificación al panel de gestión de información RDF del modelo de datos.

    2. Selecciona un modelo de la lista de éstos.

    3. Pincha en el botón de modo de edición.

    4. Elige un atributo y le asocia el estado de privacidad de "oculto " .

    5. El administrador repite el paso anterior hasta que termina de despublicar atributos del modelo y pincha en el botón "Enviar " .

    6. El sistema vuelve a mostrar el listado de atributos con su información RDF asociada en modo no edición.

- **UC-0601 Gestión de la privacidad de los datos publicados.**
  - **Descripción**: El administrador cambia la privacidad de los atributos de un modelo, de forma que será accesible con otro nivel de autorización (público o privado).
  - **Precondición**: Se debe haber generado la información de realizar la ingeniería inversa del modelo de datos para poder listarse en el panel de administración.
  - **Actores**: administrador
  - **Escenario principal**:

1. El administrador accede mediante su identificación al panel de gestión de información RDF del modelo de datos.

2. Selecciona un modelo de la lista de éstos.

3. Pincha en el botón de modo de edición.

4. Elige un atributo y le cambia el nivel de privacidad asociado al atributo en cuestión.

5. El administrador repite el paso anterior hasta que termina de establecer la privacidad a los atributos del modelo y pincha en el botón "Enviar".

6. El sistema vuelve a mostrar el listado de atributos con su información RDF asociada en modo no edición.

- **UC-0301 Consulta de los datos publicados**
  - **Descripción**: El administrador cambia la privacidad de los atributos de un modelo, de forma que será accesible con otro nivel de autorización (público o privado).
  - **Precondición**: Se debe haber generado la información de realizar la ingeniería inversa del modelo de datos para poder listarse en el panel de administración y asociada la información RDF a los modelos.
  - **Actores**: usuario externo
  - **Escenario principal**:

    1. El usuario externo realiza una consulta a través de una URI al sistema.

    2. El sistema interpreta la URI y genera la consulta SQL correspondiente.

    3. La respuesta a la solicitud se codifica en un archivo XML generado a partir de la información RDF asociada a la respuesta obtenida.

    4. El usuario recibe el XML de respuesta y procede a interpretarlo para obtener la información solicitada.

- **UC-0302 Consulta de la información de los datos publicados**
  - **Descripción**: El usuario desea consultar la información publicada del proyecto y como se realizan las consultas a éstos.
  - **Precondición**: Se debe haber generado la información de realizar la ingeniería inversa del modelo de datos para poder listarse en el panel de administración y asociada la información RDF a los modelos.
  - **Actores**: usuario externo y administrador
  - **Escenario principal**:

    1. El usuario accede al panel de información de datos publicados por el proyecto en cuestión.

    2. Visualiza la información publicada y el modo de acceso a ella.

    3. Tras realizar la consulta el usuario abandona esta interfaz.

- **UC-0801 Integración de la información RDF asociada al modelo en plantillas HTML generadas por el sistema (RDFa)**
  - **Descripción**: El desarrollador esta desarrollando una plantilla del sistema y desea integrar junto a los datos de un modelo su información RDF para generar información RDFa.
  - **Precondición**: Se debe haber generado la información de realizar la ingeniería inversa del modelo de datos para poder listarse en el panel de administración y asociada la información RDF a los modelos.

- **Actores**: desarrollador

- **Escenario principal**:

  1. El desarrollador esta realizando las plantillas que renderizan los resultados de las consultas de los usuarios de la aplicación y desea integrar la información RDF del modelo que esta mostrando.

  2. El desarrollador invoca a las funciones de consulta de información RDF y esta devuelve el resultado que se incrusta en código HTML que se generará.

  3. Se repite el paso 2 hasta que se finaliza el desarrollo de la plantilla y se integra toda la información necesaria para generar la información RDFa completa a la respuesta que representa la plantilla.

3.4.2. Modelo conceptual de datos

Aquí se muestra el modelo conceptual de datos. Este proyecto trata de una ampliación de proyectos desarrollados sobre Ruby on Rails, de aquí la simpleza que presenta su modelo conceptual frente al que puede tener una aplicación completa que posea una gestión de usuarios, sesiones, gestión de elementos propios de la aplicación, y demás funcionalidades que pudiese presentar.

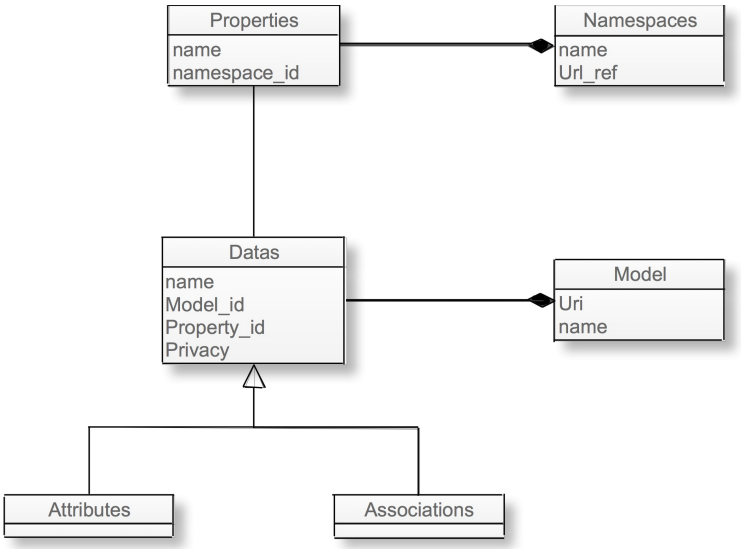


Figura 3.9: *Modelo conceptual de datos.*

3.4.3. Modelo de comportamiento del sistema

Diagrama de secuencia del sistema

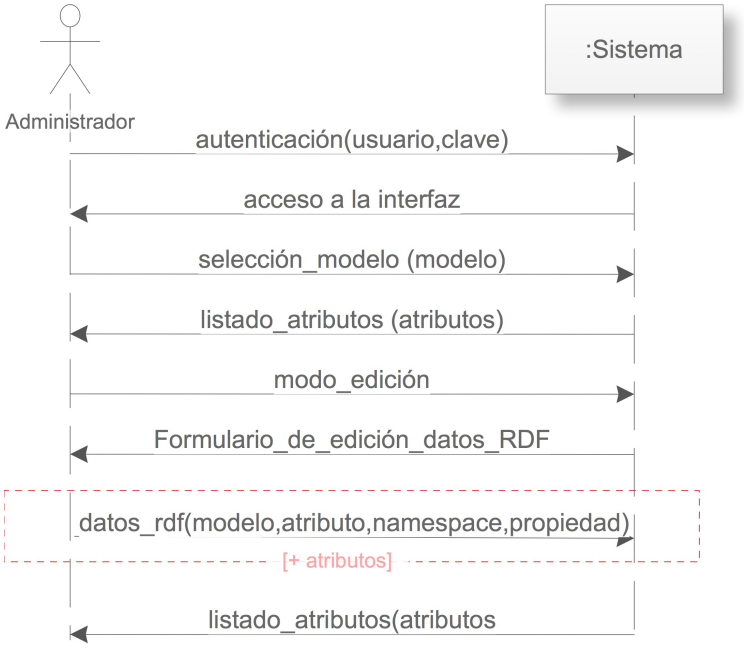


Figura 3.10: *Diagrama de Secuencia UC-0201.*

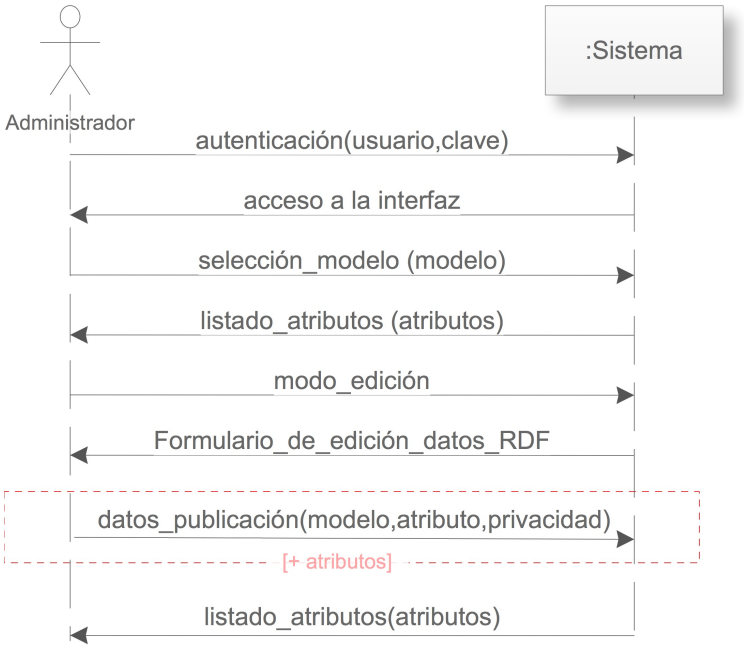


Figura 3.11: *Diagrama de Secuencia UC-0501.*

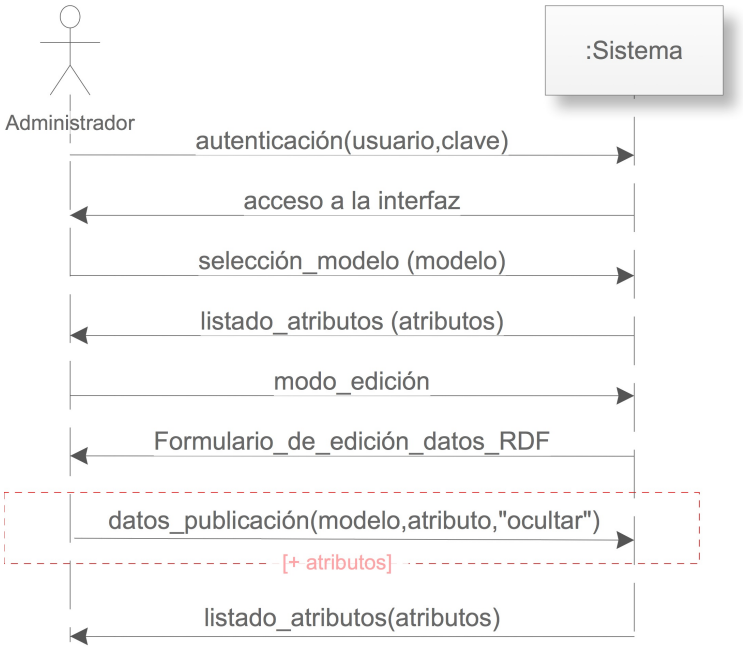


Figura 3.12: *Diagrama de Secuencia UC-0502.*

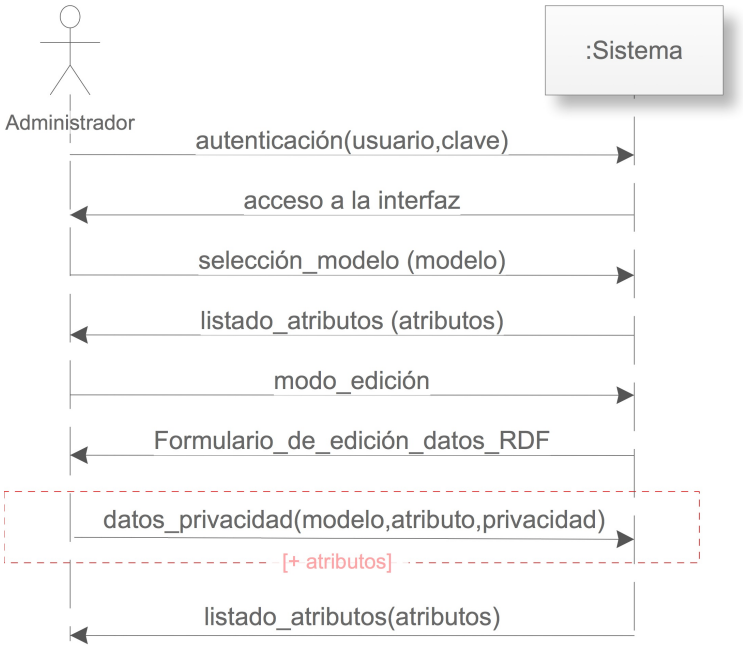


Figura 3.13: *Diagrama de Secuencia UC-0601.*

Figura 3.14: *Diagrama de Secuencia UC-0301.*

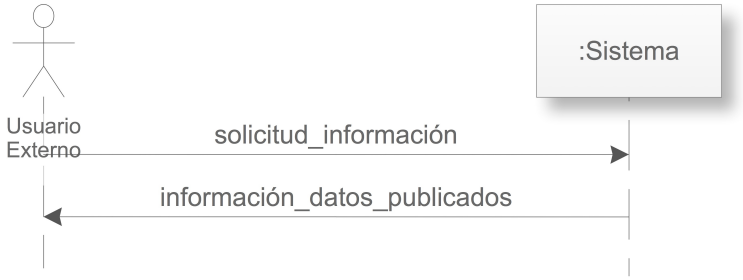


Figura 3.15: *Diagrama de Secuencia UC-0302.*

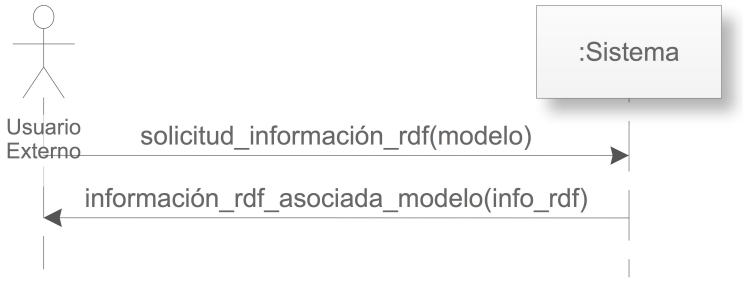


Figura 3.16: *Diagrama de Secuencia UC-0801.*

Contrato de las operaciones del sistema

- **Operación**: acceder(usuario,clave)

  Responsabilidades: Solicitar acceso a la interfaz de gestión de la información RDF del modelo de datos.

  Referencias cruzadas: UC-0201, UC-0501, UC-0502, UC-0601

  Precondiciones:

  Postcondiciones:

  - El sistema redigirá al usuario al panel de configuración.
  - En caso de ser errónea la autenticación, mostrará un mensaje indicando el error de autenticación y solicitará el envío de los datos de nuevo.

- **Operación**: seleccionar_modelo(modelo)

  Responsabilidades: Solicitar los datos del modelo, a saber, los atributos junto con sus configuraciones.

Referencias cruzadas: UC-0201, UC-0501, UC-0502, UC-0601

Precondiciones: Se ha generado la información resultado de realizar la operación de ingeniería inversa al modelo de datos.

Postcondiciones: Devuelve el listado de atributos del modelo con sus propiedades.

- **Operación**: modo_edición(modelo)

  Responsabilidades: Solicita el modo de edición de un modelo.

  Referencias cruzadas: UC-0201, UC-0501, UC-0502, UC-0601

  Precondiciones: —

  Postcondiciones: Devuelve el formulario de edición de la información del modelo en cuestión.

- **Operación**: datos_rdf(modelo, atributo, namespace, propiedad)

  Responsabilidades: Almacena los datos RDF asociados a un atributo

  Referencias cruzadas: UC-0201

  Precondiciones: Se ha generado la información resultado de realizar la operación de ingeniería inversa al modelo de datos.

  Postcondiciones: Almacena la información de *namespace* y "propiedad" asociadas al atributo del modelo de dato en cuestión.

- **Operación**: datos_publicación(modelo,atributo,privacidad)

  Responsabilidades: Publica un atributo de un modelo para su consulta por usuarios externos.

  Referencias cruzadas: UC-0501

  Precondiciones: El atributo no se encuentra publicado.

  Postcondiciones:

  - El atributo pasa a ser visible en las consultas de los usuarios externos que cumplan los requisitos de la privacidad asociada.
  - En caso de que el atributo se encontrase ya publicado, no realiza ninguna operación.

- **Operación**: datos_publicación(modelo, atributo,'ocultar")

  Responsabilidades: Despublicación de atributos de un modelo.

  Referencias cruzadas: UC-0502

  Precondiciones: El atributo se encuentra publicado.

  Postcondiciones:

  - El atributo pasa a ser oculto para las consultas de los usuarios externos.
  - Si el atributo estaba ya oculto, no realiza ninguna operación.

- **Operación**: datos_privacidad(modelo, atributo, privacidad)

  Responsabilidades: Establece un nivel de privacidad para el atributo del modelo.

  Referencias cruzadas: UC-0601

  Precondiciones: Se ha generado la información resultado de realizar la operación de ingeniería inversa al modelo de datos.

Postcondiciones: Se asocia la privacidad indicada al atributo del modelo en cuestión

- **Operación**: petición_datos(modelo,parámetros)

  Responsabilidades: Realiza una petición de datos sobre un modelo

  Referencias cruzadas: UC-0301

  Precondiciones: Se ha generado la información resultado de realizar la operación de ingeniería inversa al modelo de datos y asociado la información RDF.

  Postcondiciones:

  - Devuelve el resultado de la petición en formato XML basado en la información RDF asociada a los datos de la respuesta.
  - En caso de no producir respuesta devuelve en formato XML una respuesta de error.

- **Operación**: solicitud_información

  Responsabilidades: Solicita la información acerca de los datos publicados del proyecto.

  Referencias cruzadas: UC-0302

  Precondiciones: Se ha generado la información resultado de realizar la operación de ingeniería inversa al modelo de datos.

  Postcondiciones:

  - Muestra la información acerca de los datos publicados y el modo de acceso a éstos.
  - En caso de no cumplirse las precondiciones se mostrará un mensaje de aviso de que aun no se ha generado la publicación de datos.

- **Operación**: información_rdf_asociada_modelo(modelo)

  Responsabilidades: Solicita la información RDF asociada a un modelo concreto.

  Referencias cruzadas: UC-0801

  Precondiciones: Se ha generado la información resultado de realizar la operación de ingeniería inversa al modelo de datos y asociado la información RDF.

  Postcondiciones:

  - Devuelve la información RDF asociada al modelo.
  - En caso de no cumplirse las precondiciones se devolverá un valor vacío.

3.5. Diseño del sistema

Siguiendo con el enfoque orientado a objetos, se va a proceder a realizar el diseño del sistema *software* que dará una solución a los problemas y requerimientos hasta ahora descritos.

3.5.1. Arquitectura del sistema

A continuación se muestra un esquema de la arquitectura del sistema y su papel dentro de un proyecto anfitrión. El *software* pertenece al conjunto de *gemas* del que hará uso el proyecto Ruby on Rails. Estas *gemas* pueden estar empaquetadas en el propio proyecto o ser instaladas en el sistema junto al entorno de desarrollo Ruby on Rails.

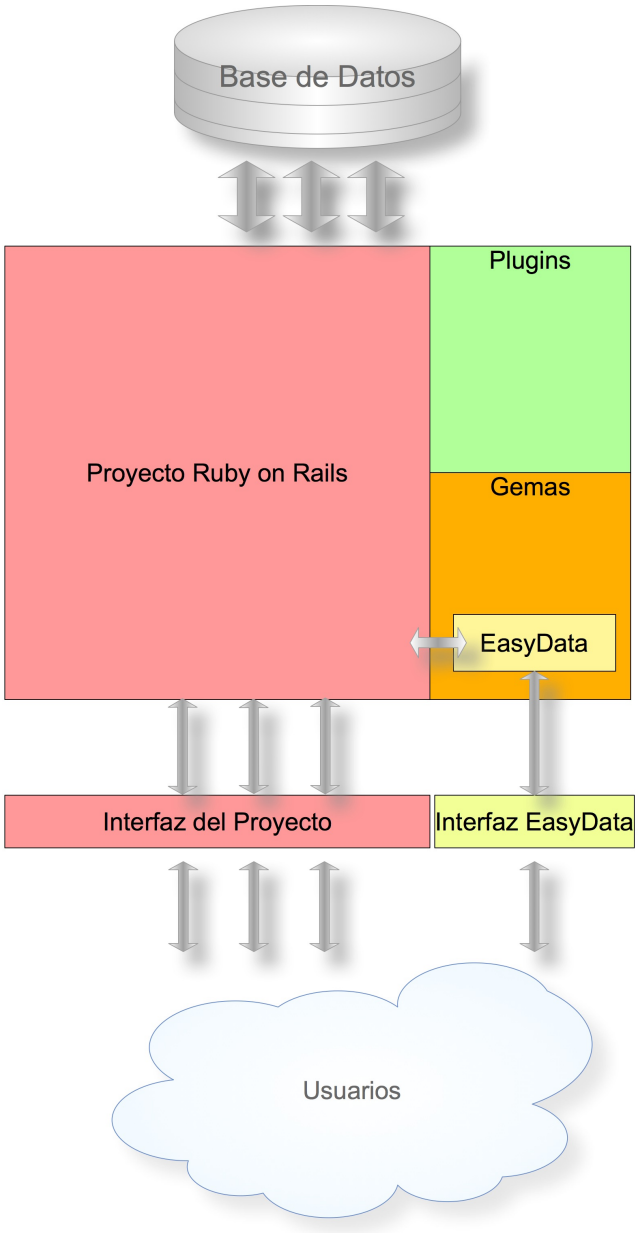


Figura 3.17: *Arquitectura* software.

En el esquema podemos apreciar como la *gema* aporta una nueva interfaz al proyecto independiente de la que se ha desarrollado para sus usuarios que permite el acceso a los datos.

3.5.2. Diseño de la capa de gestión de datos

Diseño conceptual

- Entidades.
  Las entidades representan cada una de los elementos contenidos en el modelo conceptual de datos.

Entidad	Descripción	Tipo	Atributos	Identificador
Modelo	Modelos pertenecientes al proyecto.	Fuerte	nombre, uri	nombre
Dato	Dato gestionado o propio del modelo.	Débil	nombre, modelo_id, propiedad_id, privacidad	nombre modelo_id
Atributo	Tipo de dato gestionado por el modelo.			
Asociación	Tipo de dato propio del modelo			
Namespace	Namespaces pertenecientes a RDF.	Fuerte	nombre, url_ref	nombre
Propiedad	Propiedades que componen los namespaces de RDf.	Débil	nombre, namespace_id	nombre namespace_id

Cuadro 3.4: *Entidades del sistema*

- Relaciones.
  A continuación, la tabla recoge todas las relaciones establecidas entre las distintas entidades del modelo conceptual de datos.

Relación	Descripción	Entidades
Se compone de	Asocia las propiedades con su namespace	Namespace(1,n) Propiedad(1)
Es un	Asocia un dato del modelo con una propiedad	Dato(1) Propiedad(1)
Tiene	Asocia un modelo con sus atributos	Modelo(1,n) Atributos(1)
Tipo de	Especializa la entidad dato en atributo	
Tipo de	Especializa a la entidad dato en asociación	

Cuadro 3.5: *Relaciones del sistema*

- Atributos.

- Modelo.
  Los modelos recogen la información de privacidad y RDF asociada a cada modelo de la aplicación anfitriona de la *gema* EasyData.

Nombre	**Descripción**	**Tipo atributo**	**Tipo dato**	**Dominio**	**Nulo**	**Clave**
nombre	Nombre del modelo	simple	caracter	alfanumérico	No nulo	PK
uri	URI asociada al modelo	simple	caracter	alfanumerico	No nulo	
propiedad_id	Propiedad RDF asociada al dato	simple	caracter	alfanumérico		FK
privacidad	Privacidad del atributo	simple	entero	numérico	No nulo	

Cuadro 3.6: *Atributos de la entidad Modelo*

- Dato.
  Dato representa a los atributos y asociaciones de los modelos de datos del proyecto anfitrión.

Nombre	**Descripción**	**Tipo atributo**	**Tipo dato**	**Dominio**	**Nulo**	**Clave**
nombre	Nombre del dato asociado al modelo.	simple	caracter	alfanumérico	No nulo	PK
modelo_id	Identificador del modelo al que pertenece	simple	carácter	alfanumério	No nulo	PK / FK
propiedad_id	Propiedad RDF asociada al dato	simple	caracter	alfanumérico		FK
privacidad	Privacidad del atributo	simple	entero	numérico	No nulo	

Cuadro 3.7: *Atributos de la entidad Dato*

- Namespace.
  Cada uno de los *namespace* que ofrece la *gema* para asignar información RDF al modelo o a sus datos.

Nombre	**Descripción**	**Tipo atributo**	**Tipo dato**	**Dominio**	**Nulo**	**Clave**
nombre	Nombre del namespace	simple	caracter	alfanumérico	No nulo	PK
url_ref	Url de la referencia del namespace	simple	caracter	alfanumérico	No nulo	

Cuadro 3.8: *Atributos de la entidad Namespace*

- Propiedad.
  Las propiedades componen los namespaces RDF anteriormente descritos.

Nombre	Descripción	Tipo atributo	Tipo dato	Dominio	Nulo	Clave
nombre	Nombre de la propiedad	simple	carácter	alfanumérico	No nulo	PK
namespace_id	Namespace al que esta asociada la propiedad	simple	caracter	alfanumérico	No nulo	PK / FK

Cuadro 3.9: *Atributos de la entidad Propiedad*

- Restricciones.

1. No pueden existir dos modelos con el mismo nombre.
2. Debido a la restricción 1, no pueden existir dos modelos con la misma uri.
3. No pueden existir dos datos asociados a un mismo modelo con igual nombre.
4. No pueden existir dos namespaces con igual nombre.
5. No pueden existir dos namespaces con igual url_ref (url de referencia).
6. No pueden existir dos propiedades asociadas a un mismo modelo con igual nombre.

Diseño lógico

Modelo(nombre, uri)

Dato(nombre,modelo_id,propiedad_id,privacidad)

Atributo()

Asociación()

Namespace(nombre,url_ref)

Propiedad(nombre,namespace_id)

Diseño físico

El almacenamiento de los datos físicos se va ha realizar sobre ficheros en formato YAML. Para generar la información inicial del sistema se va a utilizar un *script* que realiza una ingeniería inversa del modelo de datos y recopila la información de los distintos modelos y sus atributos y asociaciones para almacenarla en éste fichero. Inicialmente, al no tener aun información asociada, los atributos se inicializan de la siguiente forma:

1. Si el atributo contiene "_id", se trata de una clave foránea la cual servirá para construir los Linked Data pero no se publican como atributo normal. Luego no se podrán gestionar y serán ocultos a los usuario externos.

2. El resto de atributos se inicializan con ámbito "público" y con propiedad "namespace" y "propiedad" ambas iniciadas con el valor "Sin asignar"

3. Igual que los atributos anteriores, ocurre con las asociaciones del modelo.

Con esta configuración inicial estamos en disposición de poder realizar las operaciones de lectura y escritura del archivo que supondrán la ejecución de las funcionalidades *UC-0201*, *UC-05xx* y *UC-0601*, es decir, la gestión de la información necesaria para la interfaz de EasyData.

3.6. Codificación

En este apartado explicaremos las partes principales de la *gema* EasyData que se encargan de implementar las principales funcionalidades.

3.6.1. Ingeniería inversa

La primera operación que EasyData debe realizar sobre el proyecto en el que ha sido instalada, es generar el mapa de los modelos de datos de este proyecto. Para ello, mediante la función que se mostrará a continuación recorrerá los ficheros que describen los distintos modelos del sistema y obtendrá sus nombres de manera que, obtengamos su información posteriormente consultando sus definiciones.

La siguiente función recorre, como antes se ha indicado, los ficheros que contienen los modelos de datos y devuelve una lista de éstos:

```ruby
module DataModels
  # This method reads all models that the project hash.
  # @return [Array] The proyect models list.
  def self.load_models

     models = []   #All models will be in this list.
     models_valids = [] #This list is only to models with database table associated.
     mod = nil

     #Get all models in Model's folder
     Dir["#{RAILS_ROOT}/app/models/**/*.rb"].each do |file|
       models << file.gsub(RAILS_ROOT+'/app/models/',"").gsub('.rb','').classify
     end

     # Here, get the correct model's name: Singular or Plural
     models.each do |model|
       begin
         mod = eval model
       rescue
         begin
           mod = eval model.pluralize
         rescue
           mod = nil
         end
       end
       if mod
         models_valids << mod.to_s
       end
     end

     models_valids
  end

```

Figura 3.18: *Función recolectora de modelos del proyecto anfitrión.*

Una vez obtenido el listado de los modelos, procedemos a generar la información de éstos que posteriormente podremos gestionar asignándoles información RDF. Para ello, extraeremos de cada modelo sus atributos y todas sus asociaciones con otros modelos. Para obtener la definición de los atributos del modelo realizamos la siguiente operación:

```ruby
  # This method return all model's attributes.
  # @param [String] model's name
  # @return [Array] model's attribute list.
  def self.model_attributes(model)
      model.columns.map{|att| att.name}.join(", ")
  end
```

Figura 3.19: *Obtención de la información de los atributos de un modelo*

A continuación, obtenemos sus asociaciones con otros modelos mediante la siguientes lineas:

```ruby
40
41  # This method return all model's relations
42  # @param [String] model's name
43  # @return [Hash] hash with all relations and relations type
44  def self.model_relations(model)
45
46    relations = {}
47
48    #Build the hash with relation name and relation's type.
49    model.reflections.each do |relation,values|
50      relations[relation] = {"model" => values.class_name,"type" => values.macro.to_s}
51    end
52
53    relations
54  end
```

Figura 3.20: *Obtención de la información de las relaciones de un modelo.*

Una vez generada la ingeniería inversa al modelo de datos, almacenamos ésta información junto con unos valores iniciales de privacidad e información RDF en un fichero YAML que será el encargado de guardar la información RDF y de privacidad de los distintos modelos de datos del proyecto. Para el almacenamiento necesitamos del uso de la *gema* homónima para el manejo de ficheros YAML. En el siguiente apartado se muestra como se almacena y gestiona esta información mediante el uso de archivos *.yaml*.

3.6.2. Almacenamiento y gestión de la información RDF

En este apartado, explicaré como gestiona la entrada y salida de datos en archivos YAML que almacenan la información del modelo de datos y su información RDF asociada. Ruby on Rails usa este tipo de archivo tanto para almacenar información de la configuración de sus proyectos hasta para contener información de prueba para realizar el testeo de la aplicación. Por ello, para no comprometer el funcionamiento de EasyData compatible con el sistema de almacenamiento usado por la aplicación que, bajo mi experiencia, puede verse implementado bajo una complicación mayor en casos muy exclusivos. Se ha decidido el uso de éste tipo de archivos común en Ruby on Rails y que permitirá trabajar abstrayendo a la *gema* del sistema de base de datos usado en el proyecto instalada.

Una vez explicada la decisión de usar este tipo de archivos, se procede a mostrar el método de trabajo con éstos. A continuación, se muestra un ejemplo de escritura que, en este caso, se trata de la información inicial del proyecto que se genera en la instalación.

```ruby
23
24  def self.yaml_description_model(model_data)
25
26    attributes = {}
27    associations = {}
28
29    model_data.columns.each do |att|
30
31     if att.primary
32       attributes[att.name] = {:privacy => 'Private',:namespace => 'not defined',:property => 'not defined'}
33     elsif att.name =~ /_ids/
34       attributes[att.name] = "no publication"
35     else
36       attributes[att.name] = {:privacy => 'Public',:namespace => 'not defined',:property => 'not defined'}
37     end
38
39    end
40
41    model_data.reflections.keys.each do |ref|
42      associations[ref.to_s] = {:privacy => 'Public',:namespace => 'not defined',:property => 'not defined'}
43    end
44
45    {:privacy => "Private",:namespace => "not defined",:property => "not defined","attributes" => attributes,"associations" => associations}
46  end
```

Figura 3.21: *Función generadora de los datos RDF iniciales.*

En el siguiente bloque de código, se muestra la recuperación y almacenamiento en un objeto de la información almacenada en el archivo de información RDF. Este manejo de datos sobre los archivos YAML se basa en el patrón DAO (*Data Access Object*) que ofrece una abstracción en la capa presentación de la *gema* de todo este proceso y, a su vez, también integra las funcionalidades del patrón DTO (*Data Transfer Object*) para la transferencia de los datos desde la interfaz de usuario y el DAO para su actualización y/o almacenamiento. Gestionando mediante un modelo, el manejo de ésta información se hace muy sencilla para

inserción en formularios y plantillas que serán consumidas por el administrador y terceros usuarios de la interfaz ofrecida por EasyData.

```
17    #Read the document of attributes relations with rdf properties
18    # @return [Object] return a ModelRdf's instance whit rdf info
19    def initialize
20      @model_rdf = YAML::load(File.open("#{RAILS_ROOT}/config/easy_data/rdf_info.yaml"))
21    end
22
```

Figura 3.22: *Acceso a la información almacenada en el fichero* rdf_info.yml.

Estos dos ejemplos ilustran las entradas y salidas en archivos YAML que realiza la *gema* de la información RDF y configuraciones de acceso e información básica del proyecto.

3.6.3. Generación de grafos de los Linked Data

La generación de grafos que representan los Linked Data del proyecto permite visualizar las relaciones de los distintos modelos del proyecto. De forma visual, el usuario puede conocer los distintos modelos que extienden la información de un modelo en cuestión y así tiene un mayor conocimiento de la información publicada y que puede consumir del proyecto.

Esta generación se realiza mediante la librería de gráficos *Graphviz*. Ésta permite transformar un archivo *.dot* que contiene la relación entre nodos y aristas en una imagen del formato que deseemos: jpg-jpeg, png, svg, etc... De esta forma, tenemos un grafo por cada modelo que expresa sus relaciones con el resto almacenadas en la carpeta *public/images/easy_data* del proyecto Rails que integra EasyData.

A continuación, se muestra la clase que se encarga de generar estas imágenes.

```
54    def self.generate_graph(graph_info,model)
55       model = model.gsub("::","_")
56       file = File.open("#{RAILS_ROOT}/public/images/linked_data_graphs/linked_data_#{model}.dot",'w')
57
58       file.puts "digraph G {"
59       #file.puts 'size="15,15";'
60       file.puts 'graph [rotate=0, rankdir="LR"]'
61       file.puts 'node [color="#333333", style=filled,shape=box, fontname="Trebuchet MS"]'
62       file.puts 'edge [color="#666666", arrowhead="open", fontname="Trebuchet MS", fontsize="11"]'
63       file.puts model.to_s+'  [fillcolor="#116611", fontcolor="white"]'
64
65       #Draw nodes
66
67       graph_info[:assoc].each do |assoc|
68          file.puts assoc.to_s+' [fillcolor="#294b76", fontcolor="white"]'
69          file.puts '"'+model+'" -> "'+assoc.to_s+'"[dir=none];'
70       end
71
72
73       file.puts "}"
74
75       file.close
76
77       system "dot -Tpng #{RAILS_ROOT}/public/images/linked_data_graphs/linked_data_#{model}.dot -o #{RAILS_ROOT}/public/images/linked_data_graphs/linked_data_#{mod
  el}.png"
78       system "rm #{RAILS_ROOT}/public/images/linked_data_graphs/linked_data_#{model}.dot"
79    end
```

Figura 3.23: *Función generadora de los grafos Linked Data.*

Esta generación se realiza en la tarea de instalación de la *gema* que se explica en el manual de instalación.

3.6.4. Generador de RDFa

El generador de información RDFa permite generar etiquetas HTML con información RDF incrustada.

Para explicar el funcionamiento, primero debemos decir que se han tenido en cuenta en la generación de etiquetas HTML aquellas que son auto-cerradas, es decir, que acaban su declaración componiéndose de 1 sola etiqueta. Por otro lado, se ha reservado un uso especial para las etiquetas de listado *ul* y *ol* las cuales, pueden usarse para definir una instancia completa de un modelo o, si se desea, también puede manipularse igual que el resto de etiquetas.

Teniendo esto, vamos a describir alguno de los posibles usos del constructor de etiquetas HTML con información RDFa.

- Queremos listar una instancia del modelo Usuario cuyo atributo *id* es igual a 1 y que obtenga el estilo de la clase *user*.
  Podemos usar la etiqueta *ul* para este propósito, en tal caso, solo debemos escribir:

```
1.- RDFa = RDFa.new
2.- @user = Usuario.find '1'
3.- RDFa.ul('Usuario',@user,nil,"class='user'")
```

  Esta operación devolverá una estructura de *ul* con los datos solicitados y su información RDFa incrustada, en la etiqueta *ul* la referente al modelo y, en las etiquetas *li* la referente a los atributos de éste.

- Mostrar sólo un atributo (por ejemplo *nombre*) concreto de una instancia de un modelo (por ejemplo, *Coche*) en un elemento de una tabla tipo *td*.

```
1.- RDFa.td("Coche",@coche,'name',"class='coche'")
```

  Esto devolverá la etiqueta *td* con el nombre de la instancia del modelo coche siguiendo el estilo marcado en la clase homónima y con la información RDF de dicho atributo.

3.6.5. Generación de URI's e interpretación

La generación e interpretación de las URI's para devolver información RDF del modelo de datos es la principal funcionalidad de la *gema*. Se encarga de asociar a cada URI una petición de información concreta y devuelve el resultado en formato RDF. Ésta constituye la base de la integración y aprendizaje para su uso es el único requerimiento que se le exige a los usuarios externos para poder usar esta interfaz. Por ello, para facilitar esta fase de aprendizaje, se ha generado la interfaz que informa de los datos publicados, los grafos Linked Data y el uso de URI todo orientado a un aprendizaje rápido de estos usuarios.
A continuación se muestra la extensión del manejador de rutas del proyecto Rails anfitrión, que permite mapear la información RDF con URI's desreferenciables.

```ruby
module EasyDataRouting
  def self.routes(map)
    map.with_options :controller => 'easy_datas' do |ed_routes|
      ed_routes.with_options :conditions => {:method => :get} do |ed_views|
        ed_views.connect 'easy_datas/custom_rdf', :action=> "custom_rdf"
        ed_views.connect 'easy_data', :action => "custom_rdf"
        ed_views.connect 'easy_datas/authenticate_user', :action => "authenticate_user"
        ed_views.connect 's/data_publications', :action => 'info_easy_data'
        ed_views.connect 'easy_datas/info_easy_data', :action => 'info_easy_data'
        ed_views.connect 'easy_datas/linked_data', :action => 'linked_data'
        ed_views.connect 'easy_datas/access_to_data', :action => 'access_to_data'
        ed_views.connect 'easy_datas/faq', :action => 'faq'
        ed_views.connect 'easy_datas/logout', :action => 'logout'
        DataModels.load_models.each do |model|
          ed_views.connect "s/#{model.gsub("::","_")}", :controller => "easy_datas",
                                                        :action => 'show',
                                                        :model => model,
                                                        :format => 'xml'
           ed_views.connect "s/#{model.gsub("::","_").pluralize}", :controller => "easy_datas",
                                                                   :action => 'show_all',
                                                                   :model => model,
                                                                   :format => 'xml'
        end
      end
      ed_routes.with_options :conditions => {:method => :post} do |ed_actions|
        ed_actions.connect 'easy_datas/model_attributes_info', :action => "model_attributes_info"
        ed_actions.connect 'easy_datas/load_linked_data_graph', :action => "load_linked_data_graph"
        ed_actions.connect 'easy_datas/model_attributes/:model', :action => 'model_attributes'
        ed_actions.connect 'easy_datas/model_attributes_edit/:model', :action => 'model_attributes_edit'
        ed_actions.connect 'easy_datas/load_properties/:block/:attribute', :action => 'load_properties'
        ed_actions.connect 'easy_datas/login', :action => 'login'
        ed_actions.connect 'easy_datas/custom_attributes/:model', :action => 'custom_attributes'
        ed_actions.connect 'easy_datas/settings', :action => 'settings'
        ed_actions.connect 'easy_datas/custom_settings', :action => 'custom_settings'
        ed_actions.connect 'easy_datas/view_settings', :action => 'view_settings'
      end
    end
  end
end
```

Figura 3.24: *Extensión de las rutas del proyecto anfitrión para contener las propias de la* gema.

Cada URI se interpreta, se asocia y se les extrae los parámetros para realizar la consulta a través del ORM y posteriormente cruzar el resultado con la información RDF y obtener la respuesta deseada por el usuario.

Cada URI contiene en primer lugar el modelo al que se va a realizar la consulta. En caso de indicar el nombre del modelo en plural, esto sería equivalente a hacer un *Select * from [model];* y devolverá todos los registros de base de datos asociados a dicho modelo. Por el contrario, si el modelo viene en singular y acompañado de una serie de parámetros, éstos irán en la parte de los condicionantes de la consulta, es decir, *Select * From [model] Where [parámetros];*.

Para obtener los parámetros y el modelo en el controlador se consulta la variable *request* que contiene toda la información de la URI usada. A continuación el código que realiza esta función y devuelve los parámetros necesarios para la consulta.

```ruby
# Extract all parameters to build the query.
# @param [Hash] request's parameters
# @return [Hahs] Conditions hash
def parser_params (parameters = nil)

   conditions = {}

   if !parameters.empty?
     parameters.each do |key,value| #Delete all elements that aren't need to query
       unless ["controller","action","method","format","model"].include?key
         conditions[key.to_sym] = value
       end
     end
   end

   return conditions

end

```

Figura 3.25: *Extracción de los parámetros de la consulta.*

4

Evaluación

4.1. Pruebas y validación

El proyecto se ha sometido a varias pruebas necesarias para verificar y certificar cada aspecto que éste contiene. Para ello, se ha utilizado un proyecto de prueba realizado en Ruby on Rails, en concreto se trata de la aplicación de gestión de proyectos *Redmine*. Se ha instalado su última versión y se le ha incorporado la gema EasyData.

Los test que se han llevado a cabo a EasyData son los siguientes:

- Pruebas de integración
- Pruebas de validación de RDF
- Pruebas de validación de RDFa

Tanto para las pruebas de validación de RDF como RDFa se han usado varias herramientas de validación, pero ninguna obtiene una relevancia a nivel global como lo puede establecer la W3C y, por tanto, son sus validadores los principales que marcarán la soluciones de las pruebas para RDF y RDFa.

4.1.1. Pruebas de integración

Las pruebas de integración se dividen en dos partes, la primera comprueba que la gema se instala e integra sus funcionalidades al proyecto anfitrión correctamente y la segunda, comprobamos que realiza las tareas necesarias para su correcto funcionamiento, es decir que obtiene correctamente la información del modelo de datos, genera la interfaz para la gestión de información RDF de los modelos, los parámetros de configuración y el acceso funciona correctamente. Una vez comprobados y verificados estos dos grupos de pruebas podemos decir que la gema se integra correctamente y validaremos esta parte del proyecto.

Validación de instalación

Para validar el proceso de instalación de la gema realizaremos la integración en varios proyectos Ruby on Rails esperando obtener en todos un resultado válido. Para esta prueba elegiremos como proyectos anfitriones a Redmine. La limitación para la prueba de este proyecto es que los sujetos de prueba usados han de usar la versión 2.x de Ruby on Rails.

Las pruebas que verificarán los siguiente aspectos de la instalación:

- Comprobación de que la gema se añade a las lista de gemas disponibles en el entorno del proyecto. Para ello ejecutamos el siguiente comando en la misma terminal donde se ha realizado la instalación:

```
> gem list
```

- Ampliación de la lista de *tasks* o tareas del proyecto anfitrión tras la instalación de EasyData, con las tareas propias de la gema. Para esta comprobación ejecutamos el siguiente comando:

```
> rake -T | grep easy_data
```

  Y comprobamos que nos devuelve la lista de tareas de la gema EasyData.

- Correcta integración de las rutas adicionales que EasyData añade al proyecto anfitrión.

```
> rake routes | grep easy_data
```

  Igual que en el caso anterior, comprobamos que el resultado de ejecutar el comando anterior no esta vacío y se listan las rutas pertenecientes a la gema EasyData.

Después de estas tres comprobaciones podemos concluir que la instalación se ha realizado correctamente y podemos dar por válida esta prueba. Ahora se muestran los resultados obtenidos tras la instalación de la gema en los entornos de los proyectos anfitriones de prueba.

Las pruebas realizadas finalizaron todas con éxito en su instalación. Podemos afirmar que el proyecto EasyData concluye la evaluación de instalación satisfactoriamente.

Pruebas de configuración inicial

Para las pruebas de configuración inicial se usarán los mismos proyectos de prueba que en el apartado de validación de instalación. En este caso comprobaremos que la gema se integra funcionalmente con el proyecto anfitrión comprobando los siguiente puntos:

- Generación de la interfaz de configuración de la información RDF del modelo de datos. Para ello comprobamos que se ha generado la interfaz accediendo a ella.
- Obtención de la información de los modelos de datos del proyecto anfitrión. Para ello accedemos al archivo *config/easy_data/info_rdf.yml*, comprobamos que existe y que se ha generado dentro la información inicial originada por la ingeniería inversa realizada al modelo de datos.
- Comprobaciones relacionadas la configuración de acceso y personalización de la interfaz. Para llevar a cabo estas pruebas verificaremos los siguientes puntos sobre el apartado *settings* de la interfaz de administración de EasyData:
  - Acceso mediante una IP no autorizada. Introduciremos una IP distinta a la que tenemos y procederemos a desconectarnos, posteriormente, volvemos a intentar acceder a la interfaz de administración y comprobamos que nos da un error 404 de página no encontrada. Esto se deberá a que la interfaz se oculta para IP no autorizadas y aumentar la seguridad de acceso a esta interfaz.
  - Control de acceso. Establecemos un usuario y contraseña en la instalación y comprobamos que el acceso a la interfaz para este usuario se realiza correctamente.
  - Personalización. Insertar nombre, email de contacto y logotipo del proyecto anfitrión y comprobar su inclusión en la interfaz de información sobre los datos públicos que ofrece EasyData.
- Generación de gráficos de Linked Data del modelo de datos. Para ello accedemos a la interfaz de información de la publicación de datos y comprobamos que el apartado de Linked Data de cada modelo tiene su gráfico con los Linked Data representados. Esto también lo podemos comprobar accediendo al archivo *public/images/linked_data_graphs/* y comprobar que las imágenes representativas de los Linked Data de todos los modelos están correctamente generadas.

Tras comprobar estos puntos podemos verificar que la configuración inicial se ha realizado correctamente. Las pruebas realizadas sobre los proyectos de prueba fueron satisfactorias en todos los casos, y se constata que la gema está validada en su proceso de integración.

4.1.2. Pruebas de validación de RDF

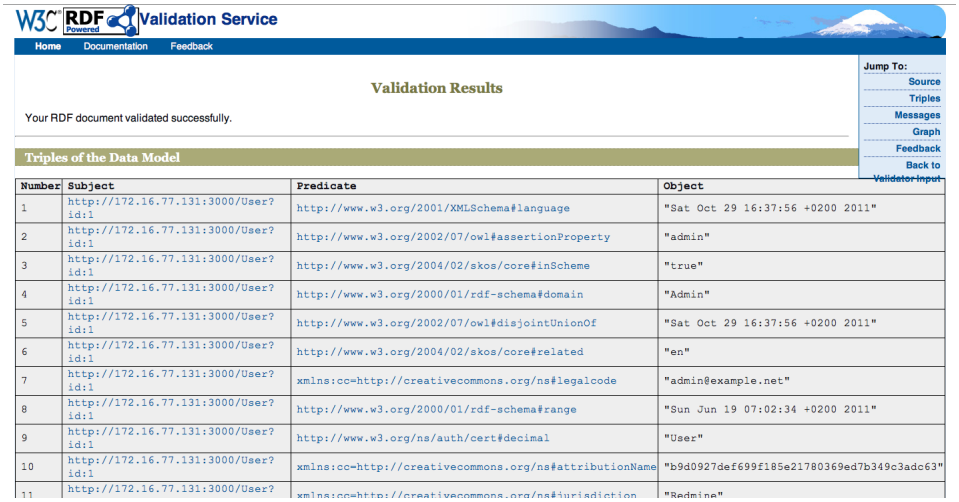
Para validar la generación de RDF en las respuestas se ha usado el validador de la W3C (http://www.w3.org/RDF/Validator/). Las pruebas consistieron en realizar a la aplicación de prueba varias peticiones mediante la interfaz de EasyData y las respuestas en formato RDF se copiaron al formulario del validador.

Las respuestas obtuvieron en su totalidad una correcta verificación por parte del validador. Las respuestas RDF generadas por EasyData están validadas por la W3C y, por tanto, podemos afirmar que están correctamente construidas y respetan las normas de la W3C para la generación de RDF.

Alguno de los ejemplos de respuestas dados por EasyData que se han utilizado para la realización de las pruebas es la solicitud de la información de un usuario de la aplicación de prueba. Aquí la respuesta en RDF dada por EasyData a dicha consulta:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
                  xmlns:cert="http://www.w3.org/ns/auth/cert#"
                  xmlns:foaf="http://xmlns.com/foaf/0.1/"
                  xmlns:skos="http://www.w3.org/2004/02/skos/core#"
                  xmlns:dc="http://purl.org/dc/elements/1.1/"
                  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
                  xmlns:owl="http://www.w3.org/2002/07/owl#"
                  xmlns:cc="xmlns:cc=http://creativecommons.org/ns#"
                  xmlns:sioc="http://rdfs.org/sioc/ns#"
                  xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
                  xmlns:wot="http://www.xmlns.com/wot/0.1/">
  <rdf:Description rdf:about="http://localhost:3000/User?id:1">
    <xsd:language>Sat Oct 29 04:32:59 +0200 2011</xsd:language>
    <owl:assertionProperty>admin</owl:assertionProperty>
    <skos:inScheme>true</skos:inScheme>
    <rdfs:domain>Admin</rdfs:domain>
    <owl:disjointUnionOf>Sat Oct 29 04:32:59 +0200 2011</owl:disjointUnionOf>
    <skos:related>en</skos:related>
    <cc:legalcode>admin@example.net</cc:legalcode>
    <rdfs:range>Sun Jun 19 07:02:34 +0200 2011</rdfs:range>
    <cert:decimal>User</cert:decimal>
    <cc:attributionName>b9d0927def699f185e21780369ed7b349c3adc63</cc:attributionName>
    <cc:jurisdiction>Redmine</cc:jurisdiction>
    <foaf:myersBriggs>all</foaf:myersBriggs>
    <sioc:subscriber_of rdf:resource="http://localhost:3000/UserPreference?id=2"/>
    <foaf:name rdf:resource="http://localhost:3000/Token?id=2"/>
  </rdf:Description>
</rdf:RDF>
```

El resultado obtenido fue el siguiente:



W3C RDF Validation Service

Home Documentation Feedback

Validation Results

Your RDF document validated successfully.

Jump To: Source Triples Messages Graph Feedback Back to Validator Input

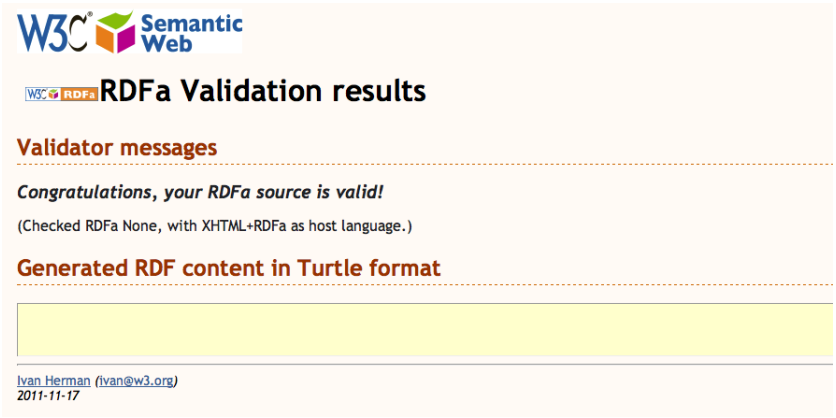
Triples of the Data Model

Number	Subject	Predicate	Object
1	http://172.16.77.131:3000/User?id:1	http://www.w3.org/2001/XMLSchema#language	"Sat Oct 29 16:37:56 +0200 2011"
2	http://172.16.77.131:3000/User?id:1	http://www.w3.org/2002/07/owl#assertionProperty	"admin"
3	http://172.16.77.131:3000/User?id:1	http://www.w3.org/2004/02/skos/core#inScheme	"true"
4	http://172.16.77.131:3000/User?id:1	http://www.w3.org/2000/01/rdf-schema#domain	"Admin"
5	http://172.16.77.131:3000/User?id:1	http://www.w3.org/2002/07/owl#disjointUnionOf	"Sat Oct 29 16:37:56 +0200 2011"
6	http://172.16.77.131:3000/User?id:1	http://www.w3.org/2004/02/skos/core#related	"en"
7	http://172.16.77.131:3000/User?id:1	xmlns:cc=http://creativecommons.org/ns#legalcode	"admin@example.net"
8	http://172.16.77.131:3000/User?id:1	http://www.w3.org/2000/01/rdf-schema#range	"Sun Jun 19 07:02:34 +0200 2011"
9	http://172.16.77.131:3000/User?id:1	http://www.w3.org/ns/auth/cert#decimal	"User"
10	http://172.16.77.131:3000/User?id:1	xmlns:cc=http://creativecommons.org/ns#attributionName	"b9d0927def699f185e21780369ed7b349c3adc63"
11	http://172.16.77.131:3000/User?	xmlns:cc=http://creativecommons.org/ns#jurisdiction	"Redmine"

Figura 4.1: *Resultado de la prueba de validación de RDF.*

4.1.3. Pruebas de validación de RDFa

Para probar la integración de información RDFa en la capa de *Vista* del proyecto de prueba, se ha generado una plantilla que muestra la información de un usuario y la información RDF asociada a cada uno de sus atributos y su descripción general. Tras renderizar la plantillas capturamos el archivo HTML generado y le pasamos el validador de marcado de la W3C (http://www.w3.org/2007/08/pyRdfa/Validator) obteniendo el siguiente resultado:



W3C Semantic Web

RDFa Validation results

Validator messages

Congratulations, your RDFa source is valid!

(Checked RDFa None, with XHTML+RDFa as host language.)

Generated RDF content in Turtle format

Ivan Herman (ivan@w3.org)
2011-11-17

Figura 4.2: *Resultado de la prueba de validación de RDFa.*

El código pasó la prueba de validación. El codigo HTML generado para la validación fué el siguiente:

```
 #http://localhost:3000/users/show_all_users

  <span class="title">All aplication's users</span>

<ul  xmls:cc=xmlns:cc=http://creativecommons.org/ns#
       xmls:rdfs=http://www.w3.org/2000/01/rdf-schema#
       xmls:skos=http://www.w3.org/2004/02/skos/core#
       xmls:dc=http://purl.org/dc/elements/1.1/
       xmls:owl=http://www.w3.org/2002/07/owl#
       xmls:cert=http://www.w3.org/ns/auth/cert#
```

```
          xmls:foaf=http://xmlns.com/foaf/0.1/
          xmls:xsd=http://www.w3.org/2001/XMLSchema#
          xmls:wot=http://www.xmlns.com/wot/0.1/
          xmls:sioc=http://rdfs.org/sioc/ns#
          xmls:geo=http://www.w3.org/2003/01/geo/wgs84_pos#
          xmls:dc=http://purl.org/dc/elements/1.1/ typeof='dc:contributor'>

  <li property='cc:jurisdiction'>7ad6b9020f7ce8bfb6f19e30f70ed899</li>
  <li property='rdfs:range'>Sun Jun 19 07:02:34 +0200 2011</li>
  <li property='cc:attributionName'>b9d0927def699f185e21780369ed7b349c3adc63</li>
  <li property='skos:inScheme'>true</li>
  <li property='skos:related'>en</li>
  <li property='dc:instructionalMethod'>1</li>
  <li property='rdfs:domain'>Admin</li>
  <li property='cc:legalcode'>admin@example.net</li>
  <li property='owl:disjointUnionOf'>Sat Oct 29 16:37:56 +0200 2011</li>
  <li property='cert:decimal'>User</li>
  <li property='cc:jurisdiction'>Redmine</li>
  <li property='foaf:myersBriggs'>all</li>
  <li property='owl:assertionProperty'>admin</li>
  <li property='not defined:not defined'>1</li>
  <li property='rdfs:isDefinedBy'></li>
  <li property='xsd:language'>Sat Oct 29 16:37:56 +0200 2011</li>

</ul>

<a href='http://localhost:3000/s/User?id=1' title='Redmine Admin' typeof='dc:contributor'>
  Redmine Admin
 </a>

<ul xmls:cc=xmlns:cc=http://creativecommons.org/ns#
      xmls:rdfs=http://www.w3.org/2000/01/rdf-schema#
      xmls:skos=http://www.w3.org/2004/02/skos/core#
      xmls:dc=http://purl.org/dc/elements/1.1/
      xmls:owl=http://www.w3.org/2002/07/owl#
      xmls:cert=http://www.w3.org/ns/auth/cert#
      xmls:foaf=http://xmlns.com/foaf/0.1/
      xmls:xsd=http://www.w3.org/2001/XMLSchema#
      xmls:wot=http://www.xmlns.com/wot/0.1/
      xmls:sioc=http://rdfs.org/sioc/ns#
      xmls:geo=http://www.w3.org/2003/01/geo/wgs84_pos#
      xmls:dc=http://purl.org/dc/elements/1.1/ typeof='dc:contributor'>

  <li property='cc:jurisdiction'>acbc2e901597abddf921c35ded60fa6a</li>
  <li property='rdfs:range'>Sat Oct 29 04:32:35 +0200 2011</li>
  <li property='cc:attributionName'>9c2dc874d6ff95e9ec7cee1b7fe9aa5f4177d3b9</li>
  <li property='skos:inScheme'>false</li>
  <li property='skos:related'>en</li>
  <li property='dc:instructionalMethod'>3</li>
  <li property='rdfs:domain'>Vazquez</li>
  <li property='cc:legalcode'>jnillo9@gmail.com</li>
  <li property='owl:disjointUnionOf'>Sat Oct 29 04:33:19 +0200 2011</li>
  <li property='cert:decimal'>User</li>
  <li property='cc:jurisdiction'>Juan </li>
  <li property='foaf:myersBriggs'>only_my_events</li>
  <li property='owl:assertionProperty'>jnillo</li>
  <li property='not defined:not defined'>1</li>
  <li property='rdfs:isDefinedBy'></li>
  <li property='xsd:language'></li>

</ul>

 <a href='http://localhost:3000/s/User?id=3' title='Juan  Vazquez' typeof='dc:contributor'>
   Juan  Vazquez
 </a>
```

Este código es un ejemplo de los generados para realizar pruebas de validación de RDFa con el validador de W3C. Es un trozo del documento HTML y se trata de la parte donde se genera la lista de usuarios del sistema con su información RDFa. Se pueden apreciar como cada atributo posee meta-información que servirá para identificar y describir cada atributo de cada usuario listado en el documento.

4.2. Pruebas de la aplicación

Para probar las funcionalidades aportadas por EasyData a los proyetos Ruby on Rails se van a realizar tres pruebas, cada una va a explotar de forma distinta las funcionalidades e interfaces que EasyData integra en los proyectos RoR. Cada una de las pruebas está enfocada a un aspecto concreto de las funcionalidades de EasyData.

- Integración con una aplicación externa mediante la interfaz de consulta en RDF.
- Explotación de la información RDFa que permite incrustar EasyData en la capa Vista de los proyectos Ruby on Rails.
- Y por último, la explotación de la información publicada por distintas aplicaciones que integran EasyData bajo un indexador de datos.

Con estas pruebas podremos exponer lo que EasyData supone como valor añadido a las aplicaciones que la integran. Estos casos son ejemplos de sus posibles usos pero no los únicos, ya que, las posibilidades son muchas más y tan simples de realizar como lo es integrar una gema a un proyecto Ruby on Rails que en menos de cinco minutos ya es completamente operativa.

4.2.1. Integración de Redmine con Moodle

Para probar la integración de la interfaz de EasyData para la integración con otras aplicaciones, se realizará un módulo de Moodle a modo de prueba que integrará la información de un determinado proyecto para su consulta en un curso concreto de Moodle. Se instalará EasyData en Redmine y se publicarán todos los datos referentes a los proyectos gestionados por este, generando así la interfaz de consulta mediante RDF que permitirá realizar las consultas del proyecto que se dé de alta para cada proyecto que active el uso del modulo en cuestión dentro de Moodle.

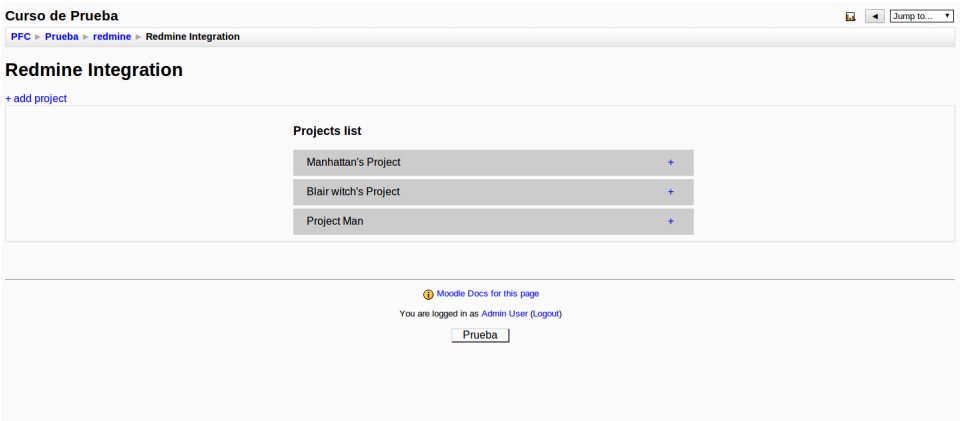


Figura 4.3: *Redmine integrado en Moodle mediante un módulo que se sirve de EasyData.*

El funcionamiento del módulo será muy trivial. Al dar de alta el modulo en un curso se pedirá las referencias para identificar el proyecto que se desea consultar. Tras esto, en el panel de herramientas lateral aparecerá el enlace a las consultas de Redmine, donde se mostrarán los distintos elementos que componen el proyecto: datos del proyecto, sus tareas, sus usuarios, etc... De esta forma tendremos integrada la información de Redmine en Moodle de una forma fácil y rápida, demostrando así las amplias posibilidades que ofrece EasyData a cualquier proyecto realizado en Ruby on Rails.

4.2.2. Explotación de la información RDFa

Usando el generador de información RDFa que ofrece EasyData, editaremos vistas de un proyecto desarrollado bajo Ruby on Rails para que contenga, en el HTML final renderizado, información RDFa que luego mediante herramientas de consulta de información RDFa se realizará una introspección para extraer la información que el documento HTML ofrece. Observaremos las ventajas aportadas por la información RDFa contenida en el código HTML generado y como los usuarios y desarrolladores pueden verse beneficiados de ella y, por supuesto, la facilidad con la que EasyData permite realizar todo esto.

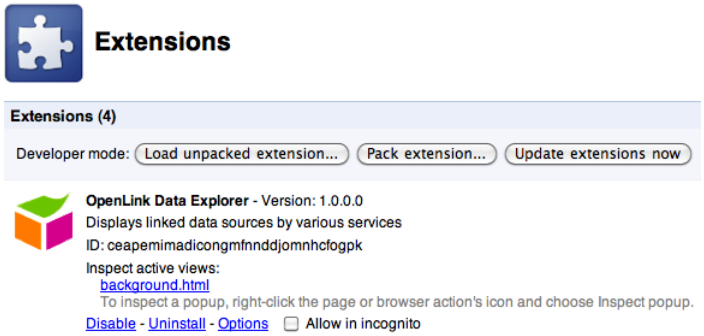


Figura 4.4: *Extensión de Firefox para la explotación de información RDFa.*

4.2.3. Indexación de información

Por último, se realizará un indexador de información que se alimenta de aplicaciones que ofrecen sus datos mediante la interfaz generada por EasyData. Esta técnica de "harvesting" o cosechadora de datos, obtendrá toda la información publica de las aplicaciones que se den de alta y permitirá realizar búsquedas de información abstrayéndonos de la aplicación que la contiene. Éste será un ejemplo del desarrollo que seguirán los buscadores en la Web Semántica y podremos ver como herramientas como EasyData acercan a las aplicaciones actuales y futuras hacia la la Web 3.0 de una forma sencilla y con un bajo coste para su propietarios tanto en recursos como en tiempo.

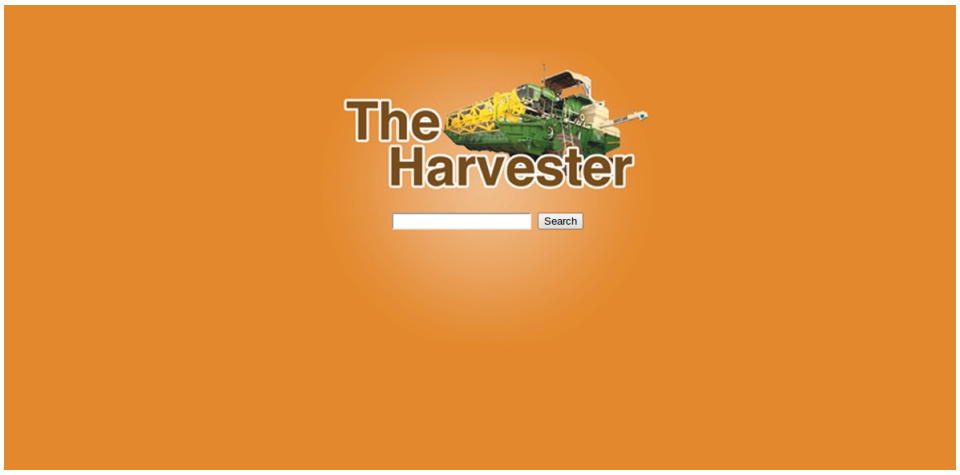


Figura 4.5: *Aplicación de indexación de información publicada con EasyData.*

5

Conclusiones

5.1. Conclusiones Generales

Como resultado del desarrollo de la *gema* EasyData podemos decir que tenemos una herramienta que será capaz de desarrollar una interfaz funcional, que permitirá la publicación de los datos del modelo de datos en aplicaciones Ruby on Rails. Esto no sólo facilitará esta interoperabilidad con otras aplicaciones externas sino que lo hará aún coste insignificante si lo comparamos con el coste que tendría realizar esta interfaz sin esta herramienta.

EasyData supone un nuevo camino en el desarrollo rápido de interfaces de comunicación y publicación de datos que supondrá la inclusión en la Web Semántica de muchas aplicaciones que, tal vez por el coste que suponía o tiempo de desarrollo, no tenían esta opción entre sus prioridades. Aumentando el número de aplicaciones que publican sus datos, podremos construir buscadores que generen una información más precisa, realizar integraciones entre aplicaciones para conseguir entornos más potentes y funcionales de cara a los usuarios finales de éstos. En definitiva, conseguir una nueva perspectiva de la Web que permitirá una generación de información y funcionalidades que el usuario final será el principal benefactor de todas ellas.

5.2. Futuros Trabajos

El camino a seguir para desarrollar futuras versiones de este *software*, tiene que ir orientado a integrar aun más los proyectos Ruby on Rails en la Web Semántica. Por ello, a continuación se lista una serie de posibles caminos de actuación para conseguir este propósito.

- Desarrollar una comunicación de dos sentidos. Esto hace referencia a que además de ofrecer información, la interoperabilidad con otras plataformas deba permitir la inserción de datos considerando las restricciones que se han indicado en el modelo de datos de la aplicación.
- Orientar la generación de información a la lógica de negocios siguiendo las normas BPEL.

De todos los trabajos futuros que puede seguir este proyecto y debido a que la base del proyecto no depende de la tecnología usada. Una vez madurada la gema y desplegado su funcionamiento en entornos desarrollados bajo Ruby on Rails, el siguiente paso será su exportación a otras tecnologías de desarrollo de aplicaciones Web como pueden ser *Symfony*, *Django* o incluso buscar una solución que permita abstraerse de la tecnología usada en los distintos proyectos donde se desee integrar, permitiendo así incluir las funcionalidades que con EasyData se llegan a ofrecer.

Esta última linea de trabajo es la más ambiciosa y la más compleja, pero sin duda, si se consiguiese desarrollar una solución con tales características, obtendríamos representado al

completo la idea fundamental y para la que se ha desarrollado este proyecto. Obtendríamos una herramienta de gran valor para conseguir que proyectos de diferentes tecnologías se introdujeran en la Web Semántica con un coste de desarrollo muy reducido tanto en tiempo como en recursos.

Sin embargo, la mejora más inmediata que se aplicará a EasyData será la posibilidad de inserción de datos. Esta mejora permitirá a usuarios registrados en el sistema y con los permisos necesarios, insertar datos en la aplicación a través de la interfaz que ofrece EasyData. Para llevar a cabo esta tarea, se generará un sistema de control de permisos de usuarios basado en la identificación mediante *API_KEY*. Esta elemento es un identificador único que el proyecto asignará a cada usuario para que puedan insertar elementos a través de la interfaz de EasyData. A continuación, se expone una posible solución al control de usuarios basado en *API_KEY* para la inserción de datos.

La idea sería establecer un control intermedio previo y posterior a las acciones realizadas en los controladores para comprobar que la petición del usuario es correcta y que éste posee los permisos necesarios para llevar la operación a cabo. Esto se consigue de la siguiente forma:

```
# app/controllers/api_controller.rb

   class ApiController < ActionController::Base
      attr_accessor :current_user
      prepend_around_filter ApiAuthorizedFilter.new
   end

# app/models/api_authorized_filter.rb

  class ApiAuthorizedFilter
    def before(controller)
       return true unless controller.params[:api_key]
       controller.current_user = User.find_by_api_key(controller.params[:api_key])
    end

    def after(controller)
      controller.current_user = nil
    end
  end
```

Con la clase *ApiController*, tenemos dos *callbacks* definidos que se ejecutarán uno antes y otro después de la acción pertinente. La primera, comprueba previamente que el usuario con la *API_KEY* indicada existe y genera el *controller.user* con los datos del usuario que esta ejecutando la acción, con esto comprobamos si tiene permisos para realizar dicha operación. En el segundo *callback* eliminamos el usuario para que no pueda realizar más operaciones al menos que vuelva a realizar una llamada con su *API_KEY* correctamente insertada en ella.

Ahora el resto de controladores deberán heredar de esta clase *ApiController* para poder utilizar la interfaz de EasyData. Por ejemplo:

```
#app/controllers/users_controller.rb

class UsersController < ApiController
     [Acciones]
end
```

De esta forma, todos los controladores poseen el paso previo que es el control de la *API_KEY* del usuario. Este ejemplo ha sido uno de las posibles implementaciones de la solución para la inserción de datos a través de la interfaz proporcionada por EasyData a los proyectos realizados en Ruby on Rails. La fuente de este ejemplo viene en la bibliografía y es un buen ejemplo del posible desarrollo ha seguir en las próximas versiones de la *gema* EasyData.

A

Publicaciones

En este anexo se incluyen las publicaciones en las que ha participado el autor relacionados con éste proyecto.

- *Open linked data model revelation and access for analytical web science*
  Juan Manuel Dodero, Iván Ruíz-Rube, Manuel Palomo Duarte y Juan Vázquez Murga.
  5º Metadata and Semantics Research Conference (MTSR-2011)

B

Manual de usuario

B.1. Introducción

El sistema tiene tres tipos de usuarios finales los cuales usarán la aplicación para dar solución a sus necesidades particulares. Por ello, se ha dividido el manual de usuario en tres partes, cada una contempla a un usuario final de la aplicación y se expondrá las distintas funcionalidades orientadas a ellos y su modo de uso.

B.2. Manual de Usuario: Administrador

Este usuario realiza sus operaciones en el sistema de forma visual, sin necesidad de acceder al código fuente del software. Sus funcionalidades se ejecutan en la interfaz de administración de sistema *software* y son las siguientes.

- **Administración de los datos publicados.**
  Para administrar los datos publicados del proyecto, debemos acceder a la interfaz de administración de información RDF del proyecto. Para ello, accedemos a la siguiente dirección:

```
> http://[host]/custom_rdf
```

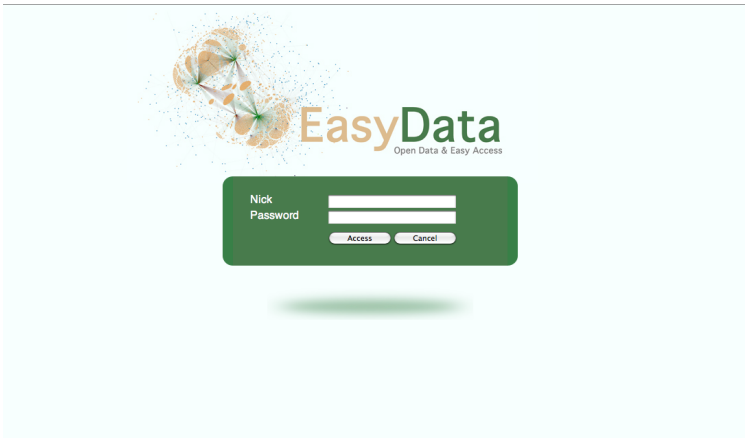


Figura B.1: *Captura de la pantalla de login*

Se nos pedirá que nos autentifiquemos para comprobar que poseemos las credenciales necesarias para llevar a cabo ésta tarea. Una vez dentro, se nos listan los modelos del

proyecto. Pinchando en el nombre de cada modelo, se desplegará la lista de atributos y asociaciones de éste y su información asociada.

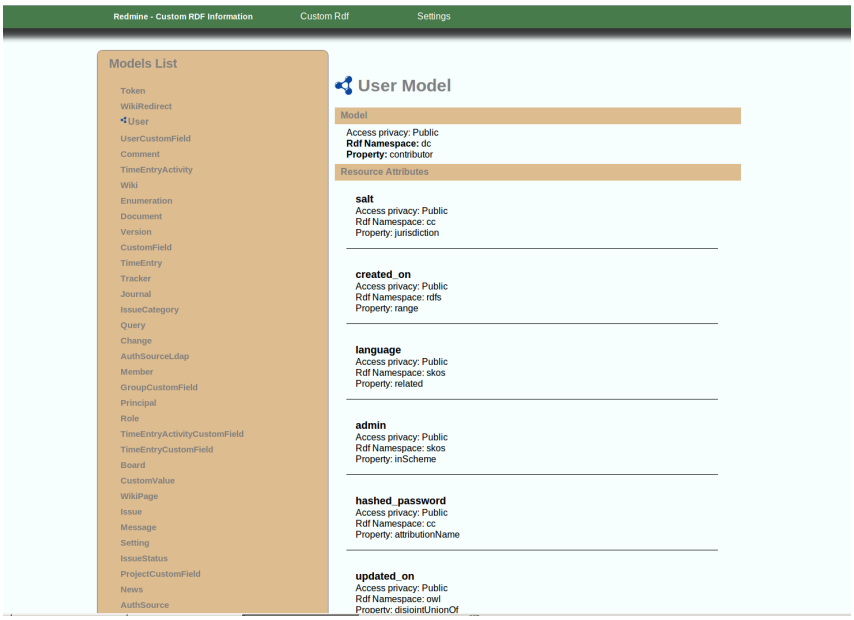


Figura B.2: *Captura de la pantalla de la edición de la información asociada al modelo de datos.*

Para administrar esta información, en la parte inferior de la lista aparece el botón "Editar" que, pinchando en él, aparecerá el formulario de edición de la información asociada a los atributos. En él podremos, hacer las siguientes tareas:

- Publicar y ocultar datos. Estableciendo su privacidad a "Hidden", ocultaremos los datos contenidos en dicho atributo y, si le asignamos cualquier otra privacidad lo publicaremos.
- Establecer la privacidad de datos. En el campo de caso anterior, podemos establecer si el dato representado por el atributo es privado o público, lo cual repercute en el estado del usuario externo para mostrarse o no, respectivamente.
- Asignar *namespace* y propiedad al atributo. De esta forma daremos una descripción basada en reglas RDF del dato contenido por el atributo.

Tras los cambios realizados, podemos guardarlos pinchando sobre el botón "Enviar" del formulario. El sistema, si no se ha producido ningún error, el sistema nos devolverá a la lista de atributos del modelo con la información actualizada.

- **Consulta de la información sobre los datos publicados.** Este usuario al igual que el usuario externo, puede comprobar el estado final de los datos publicados a través de la información asociada a éstos. Esta funcionalidad va orientada a visualizar y comprobar, por parte del administrador, el estado final de los datos publicados. Para acceder a esta información, accedemos a través de la siguiente *url*:

```
> http://[host]/info_linked_data
```

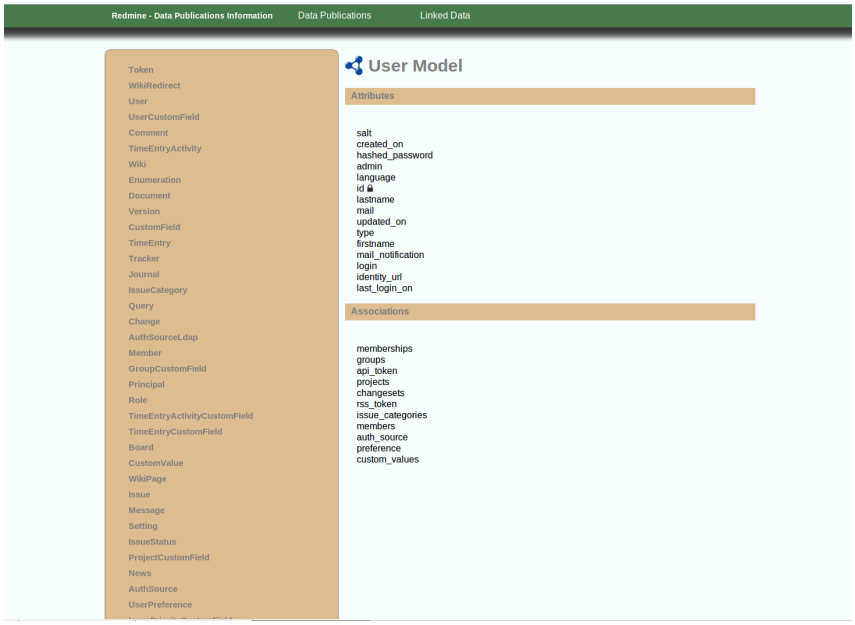


Figura B.3: *Captura de la pantalla de listado de la información publicada del modelo de datos.*

B.3. Manual de Usuario: Desarrollador

Este usuario que posee la tarea de desarrollar la capa *Vista* del proyecto, se encarga de generar la información RDFa del proyecto basándose en la información RDF asociada al modelo de datos.

- **Inclusión de la información RDFa en la capa *Vista* del proyecto.** La información RDF asociada al modelo, se puede insertar en las etiquetas HTML mediante el generador de información RDFa que dispone la gema.

B.4. Manual de Usuario: Usuario externo

- **Realizar consultas** Para realizar consultas sobre la información publicada, este usuario debe construir una URI desrefenciable siguiendo el siguiente patrón:

```
http://[host]/s/[Modelo]/[Parametros]
```

  - **host**: *host* de la aplicación.
  - **Model**: Modelo a consultar, con el formato de primera letra de cada palabra en mayúscula y sin espacios.
  - **Parámetro**: Son los parámetros de consulta, que siguen el formato [atributo1]:[valor1]&[atributo2]:[valor2]... A modo de ejemplo, vamos a construir una consulta al proyecto desplegado en local y escuchando en el puerto 3000 (el puerto por defecto usado por Ruby on Rails para desplegar) sobre el modelo Usuario Particular con nombre “Juan” y apellidos “Tenorio”:

```
> http://localhost:3000/s/User?name:Juan&apellidos:Tenorio
```

    En caso de no indicar ningún parámetro e indicar el nombre del modelo en plural, se listarán todas las instancias del modelo almacenadas.

```
> http://localhost:3000/s/Users
```

- **Consulta de la información sobre los datos publicados.** Esta funcionalidad es compartida con el usuario administrador y su modo de actuación es equivalente al citado en la parte del manual asociada al usuario administrador. Se accede mediante la siguiente URL:

```
> http://localhost:3000/s/data_publications
```

C

Manual de instalación

C.1. Requerimientos del sistema

Los requerimientos que el sistema debe tener para el correcto funcionamiento de la gema, son los siguientes:

- *Ruby* 1.8.7 ó 1.9.x
- *Rails* en su versión 2.3.x
- Gema *Hpricot* igual o mayor a la versión 0.8.4
- *Yaml*
- *Haml* en su versión o mayor a 3.0.0
- Biblioteca para la generación de graficos *Graphviz*.
- Biblioteca *ftools* para la gestión de archivos y directorios en el sistema.

C.2. Instalación

Para la instalación de gemas en un entorno Ruby, necesitamos de tener instalado el paquete *gem*, que podemos encontrar en www.rubygem.org. Una vez instalado este paquete, tendremos acceso a todo el repositorio de gemas que ofrece el repositorio *RubyGem* y, entre éstas, EasyData. Para instalarla necesitamos ejecutar el siguiente comando:

```
> gem install easy_data
```

Una vez descargada la gema de *Rubygems*, necesitamos incluirla en el proyecto. Para ello, el primer paso es insertar las rutas de la interfaz de EasyData dentro del documento de rutas del proyecto anfitrión para que dicha interfaz sea accesible. Para ello, insertamos la siguientes lineas en las rutas del proyecto que se encuentran en el archivo *config/routes.rb*:

```
# config/routes.rb

 EasyDataRouting.routes(map)
```

A continuación, en el archivo *Rakefile* del proyecto, insertamos la siguiente cabecera:

```
require "easy_data/tasks"
```

Con esta linea, conseguimos añadir las tareas de la gema EasyData al proyecto anfitrión. Estas nos harán falta en el siguiente paso para completar la configuración inicial de la gema.

C.3. Configuración inicial

Una vez instalada la gema, debemos realizar la configuración inicial necesaria para comenzar a usarla. Para ello, debemos ejecutar el *script* de instalación de EasyData en el proyecto. Este *script* realizará la ingeniería inversa del modelo y almacenará la información recopilada, copiará el archivo CSS necesario para la interfaz de configuración al proyecto y copiará al proyecto la información de inicialización requerida por la gema.

Para realizar esta configuración inicial, ejecutaremos el siguiente *script*:

```
> ruby easy_data:install RAILS_EVN=[entorno]
```

C.4. Pruebas de instalación

El contenido de esta sección está dedicado exclusivamente a comprobar que la instalación del *software* ha sido satisfactoria. Para ello se van a indicar una serie de operaciones mediante la cual podremos saber que el proceso de instalación se ha realizado correctamente.

- **Instalación de la gema.** Comprobamos que la *gema* se ha instalado correctamente y se encuentra en la lista de gemas instaladas en el servidor:

  ```
  > gem list
  ```

- **Generación de rutas para las acciones de la gema EasyData.** Para comprobar que se han añadido satisfactoriamente las ruta propias de la gema, utilizaremos el siguiente comando:

  ```
  > rake routes
  ```

- **Incrusión de las tareas.** Estas tareas se encargan de la generación de la información inicial requerida por el *software*. Para comprobar que este paso se ha realizado correctamente ejecutaremos:

  ```
  > rake -T
  ```

- **Interfaz de configuración.** Por último, comprobamos que hay acceso a la interfaz de configuración:

  ```
  http://[host]/custom_rdf
  ```

Bibliografía

[1] Michael Hausenblas/ Exploiting Linked Data to Build Web Applications.Digital Enterprise Research Institute, Galway (2009)

[2] R. Battle, E. Benson / Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST).Web Semantics: Science, Services and Agents on the World Wide Web 6 (2008) 61,69.Available online at http://www.sciencedirect.com

[3] Conferencia de Tim Berners Lee sobre "Linked Data.[en] el TED. http://linkedata.blogspot.com/2010_11_01_archive.html

[4] W3C Web Site, RDF Primer. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[5] W3C Web Site, Rdfa Primer. http://www.w3.org/TR/xhtml-rdfa-primer

[6] Learn REST: A Tutorial. http://rest.elkstein.org/2008/02/documenting-rest-services-wsdl-and-wadl.html

[7] hRests. http://knoesis.wright.edu/research/srl/projects/hRESTs/

[8] Resource (REST) programming model and conventions.http://www.projectzero.org/sMash/1.1.x/docs/zero.devguide.doc/zero.core/REST.html

[9] Christian Bizer, Richard Cyganiak, Proceedings of the 5th ISWC (2006), D2R Server – Publishing Relational Databases on the Semantic Web.

[10] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, et al. Proceedings of the 3rd International Semantic Web User Interaction Workshop (2006), Tabulator: Exploring and Analyzing linked data on the Semantic Web.

[11] Benjamin Adrian, Jörn Hees, Ivan Herman, Michael Sintek, Andreas Dengel, Knowledge Engineering and Knowledge Management by the Masses EKAW 2010 (2010), Epiphany: Adaptable RDFa Generation Linking the Web of Documents to the Web of Data

[12] Eyal Oren, Sebastian Gerke , Proceedings of the WWW Conference (2007) ,ActiveRDF: Object-Oriented Semantic Web Programming

[13] Tom Heath, Christian Bizer, Linked Data: Evolving the Web into a Global Data Space (2011)

[14] T. Berners-Lee, J. Hendler, O. Lassila, Scientific American (2001) ,The Semantic Web

[15] Michael Hausenblas, IEEE Internet Computing (2009), Exploiting Linked Data to Build Web Applications

[16] Tim Berners-Lee on the next Web conference. http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html

[17] Rails Guides, The basic of Creating Rails Plugins. http://guides.rubyonrails.org/plugins.html

[18] Plugin RDFa on Grails. http://www.grails.org/plugin/rdfa

[19] W3C RDF Validator http://www.w3.org/RDF/Validator/